The TikZ-UML package

Nicolas Kielbasiewicz

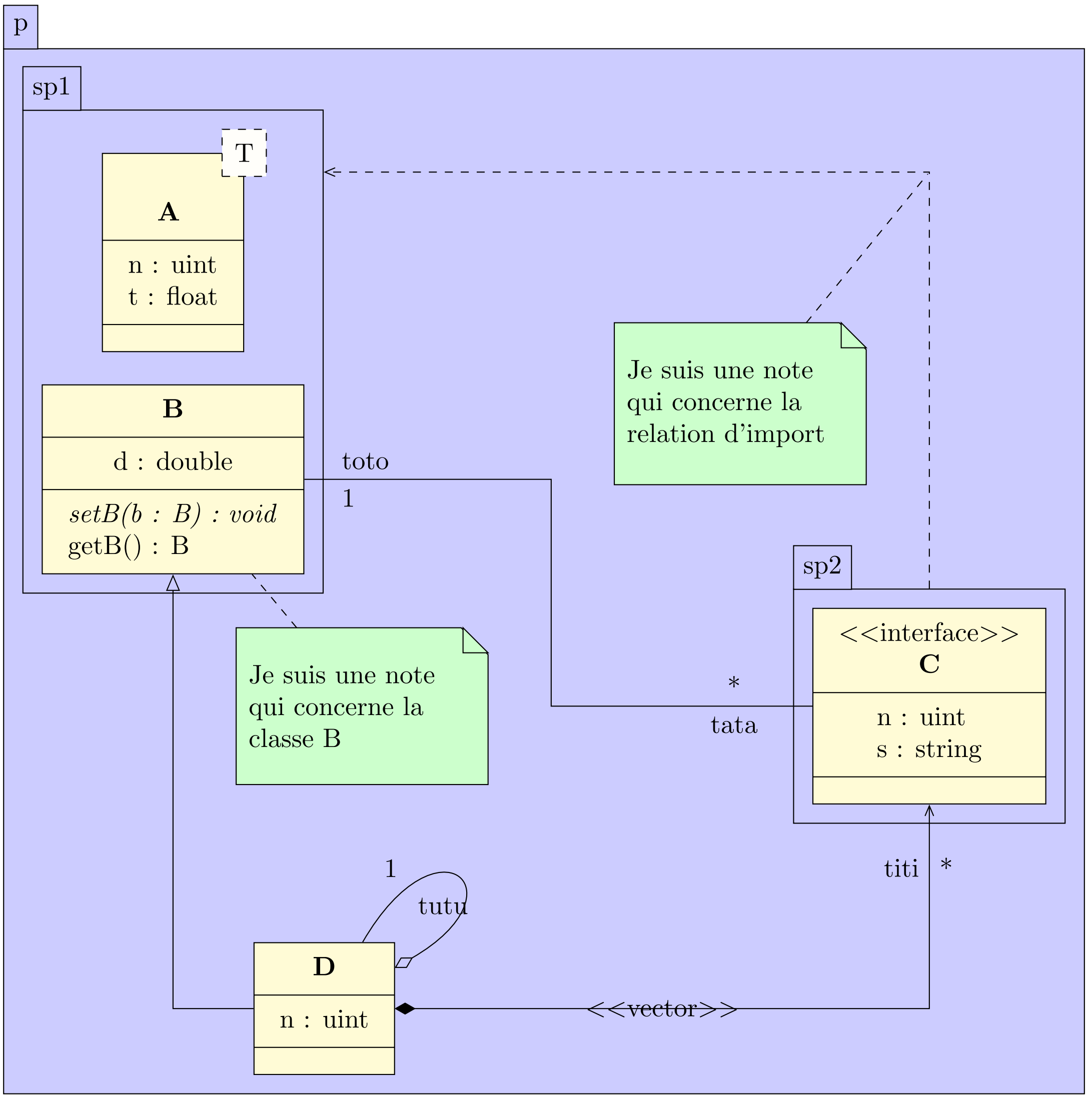
February 29, 2012

Contents

Preamble

In front of the wide range of possibilities given by the PGF/TikZ library, and in front of the apparent lack of dedicated packages to UML diagrams, I was to develop the TikZ-UML package, with a set of specialized commands and environments for these diagrams. It is dedicated to succeed pst-uml package, that was developped for similar reasons in PSTricks. Actually, the package contains definitions of complete class diagrams, use case diagrams, sequence diagrams and state diagrams in a quite easy way. Some improvements are needed, but it is near the final release.

Here is an example of class diagram you can draw :



We will now show you the various functionnalities of TikZ-UML , but before we will talk about packages dependencies and installation of TikZ-UML .

Dependencies

Required packages

tikz : It is useless to present this extremely powerful and complete drawing package. Every diagram generated by TikZ-UML is in fact generated by TikZ . It also gives some packages and libraries used by TikZ-UML , such as the backgrounds, arrows, shapes, fit, shadows, decorations.markings libraries and the **pgfkeys** package that manages macros options.

ifthen : This package offers the management of conditional test, such as if . . . then . . . else . . .

xstring : This package offers string substitutions. It is used for the management of names between programmation items (classes, states, packages, . . .) and and the nodes representing them.

calc : This package offers easy access to calculations.

pgfopts : This package is an add-on of the **pgfkeys** package for the management of packages and classes options.

Others dependencies

For still unknown reasons, TikZ-UML works fine if you have already included the **babel** package with the language of your choice.

Installation

Coming soon

Chapter 1

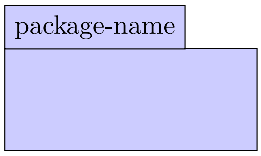
Class diagrams

1.1 Package, class, attributes and operations

1.1.1 To define a package

You can define a package with the `umlpackage` environment :

```
\begin{tikzpicture}
\begin{umlpackage}[x=0,y=0]{package-name}
\end{umlpackage}
\end{tikzpicture}
```



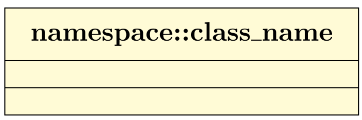
Both options `x` and `y` allow to define the package position in the figure. The default value for both of them is 0.

- When a package contains classes and sub-packages, its dimensions automatically fit to its content.
- You can define as many packages as you want in a figure.
- For an empty package (that contains no class), you can use a shortcut command : `umlemptypackage` that takes the same arguments and options as the `umlpackage` environment

1.1.2 To define a class

You can define a class with the `umlclass` command :

```
\begin{tikzpicture}
\umlclass[x=0,y=0]{namespace::class\_name
    }{}{}
\end{tikzpicture}
```

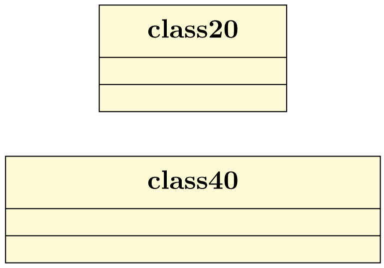


Both options `x` and `y` allow to define the class position in the figure. 2 cases : if the class is defined inside a package, the class position is relative to the package; on the contrary, the class position is relative to the figure. The default value for both options is 0. For an empty class (that contains no attributes and no operations), you can use a shortcut command `umlemptyclass` that takes only the class name for argument and the same options as the command `umlclass`.

To define the width of a class

The default width of a class is 10ex. You can use the `tt width` option to specify an other value :

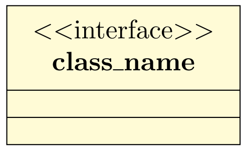
```
\begin{tikzpicture}
\umlemptyclass[width=15ex]{class20}
\umlemptyclass[y=-2, width=30ex]{class40}
\end{tikzpicture}
```



To specify the type of a class

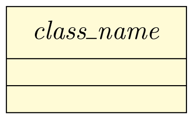
There is different types of classes : class, interface, typedef, enum. You can specify it with the `type` option (the default value is class) :

```
\begin{tikzpicture}
\umlemptyclass[type=interface]{class\_
    name}
\end{tikzpicture}
```



The type is written between $\ll$ and $\gg$ above the class name, excepted the class type (default behavior), and the abstract type, where the class name is written in italic style instead :

```
\begin{tikzpicture}
\umlemptyclass[type=abstract]{class\_name
    }
\end{tikzpicture}
```

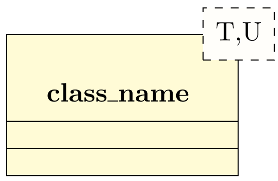


Notice that aliases exists for each value of the type option : `umlabstract`, `umltypedef`, `umlenum`, `umlinterface`.

To specify template parameters

For a template class, you can use the `template` option to specify the template parameters list :

```
\begin{tikzpicture}
\umlemptyclass[template={T,U}]{class\_
    name}
\end{tikzpicture}
```



Name of the node defining a class

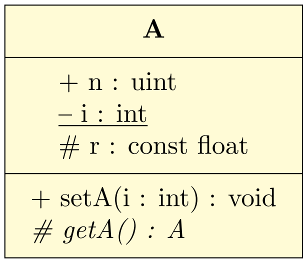
Pour donner le nom d'une classe, il arrive que l'on utilise des caracteres speciaux, comme le _ ou les : quand on specifie le namespace. Le mecanisme interne de TikZ-UML nomme le nœud definissant une classe avec le meme nom. Or, l'utilisation du \ et des : est interdite pour nommer un nœud en TikZ . Il est tout a fait possible que d'autres caracteres viennent poser probleme. Il faut donc proceder a une substitution de caractere, operation appelee des que l'on definit ou utilise le nom d'une classe. Nous avons vu dans les exemples precedents que cela fonctionne pour le _.

1.1.3 To define attributes and operations

The attributes of a class are defined with the second argument of the `umlclass` command. You write the attributes list using
as a delimiter. The operations of a class are defined with the third argument of the `umlclass` command.

To define a static attribute or a static operation, you can use the `umlstatic` command. In a similar way, the `umlvirt` command is used to define a virtual operation :

```
\begin{tikzpicture}
\umlclass{A}{
  + n : uint \\ \umlstatic{-- i : int} \\
      \# r : const float
  }{
  + setA(i : int) : void \\ \umlvirt{\#
      getA() : A}
}
\end{tikzpicture}
```

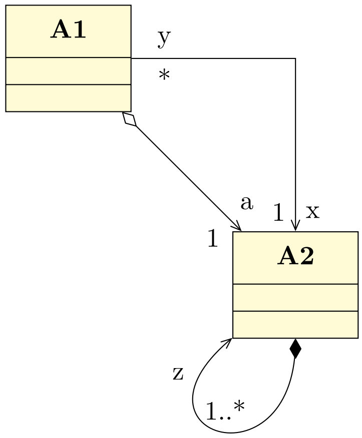


1.2 To define a relation between classes

1.2.1 General macro

Each class or package is draw as a node sharing the same name. To define a relation between two classes, you just need to specify the source class name, the target class name and a set of options specific to the relation :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=3,y=-3]{A2}
\umluniaggreg[arg2=a,mult2=1,pos2=0.9]{A
    1}{A2}
\umluniassoc[geometry=-|,arg1=x,mult1=1,
    pos1=1.9,arg2=y,mult2=*,pos2=0.2]{A1}{
    A2}
\umlunicompo[arg=z,mult=1..*,pos=0.8,
    angle1=-90, angle2=-140, loopsize=2cm
    ]{A2}{A2}
\end{tikzpicture}
```



From a UML semantic point of view, there are 11 different relations. Every type of relation is defined in TikZ-UML :

A dependency : You can use the `umldep` command

An association : You can use the `umlassoc` command

A unidirectional association : You can use the `umluniassoc` command

An aggregation : You can use the `umlaggreg` command

A unidirectional aggregation You can use the `umluniaggreg` command

A composition : You can use the `umlcompo` command

A unidirectional composition : You can use the `umlunicompo` command

An import : You can use the `umlimport` command

An inheritance : You can use the `umlinherit` command

An implementation : You can use the `umlimpl` command

A realisation : You can use the `umlreal` command

These 11 shortcuts are based on the same scheme (the `umlrelation` command) and take theoretically the same set of options. Nevertheless, some options concern only part of them.

1.2.2 To define the geometry of a relation

As you may have seen in previous examples, you can specify the geometric shape of a relation with the `geometry` option. It needs a value among the following list : - - (straight line), -| (horizontal then vertical), |- (vertical then horizontal), -|- (horizontal chicane) ou |-| (vertical chicane). These values are very inspired from TikZ philosophy.

It seems that this option is used very often. That is why a shortcut of the `umlrelation` command has been defined each possible value of the `geometry` option :

umlHVrelation : shortcut of `umlrelation` with geometry=-|

umlVHrelation : shortcut of `umlrelation` with geometry=|-

umlHVHrelation : shortcut of `umlrelation` with geometry=-|-

umlVHVrelation : shortcut of `umlrelation` with geometry=|-|

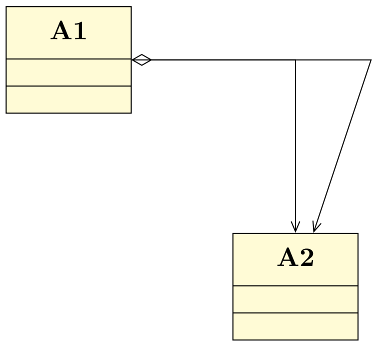
For each of these 4 shortcuts, the `geometry` option is forbidden.

There is no shortcut for the value - - : this is the default value for the `umlrelation` command.

1.2.3 To adjust the geometry of a relation

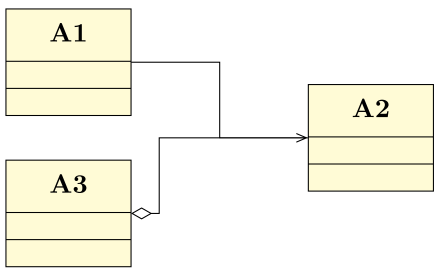
When the geometry is built with 2 segments, you can define the coordinates of the auto-built point, named control node. Then, instead of using `umlrelation`, you should use the `umlCNrelation` command, or one of its 11 shortcuts :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=3,y=-3]{A2}
\umluniaggreg[geometry=-|]{A1}{A2}
\umlCNuniassoc{A1}{4,0}{A2}
\end{tikzpicture}
```



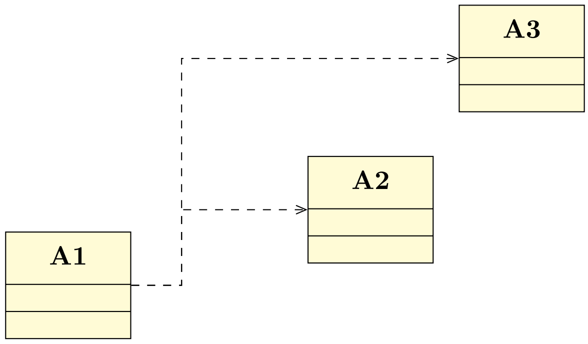
When the geometry is built with 3 segments, the relative position of the middle segment between classes is defined by the midlle of the classes nodes. You can adjust this parameter with the `weight` option :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=-1]{A2}
\umlemptyclass[y=-2]{A3}
\umlassoc[geometry=-|-]{A1}{A2}
\umluniaggreg[geometry=-|-, weight=0.3]{A
   3}{A2}
\end{tikzpicture}
```

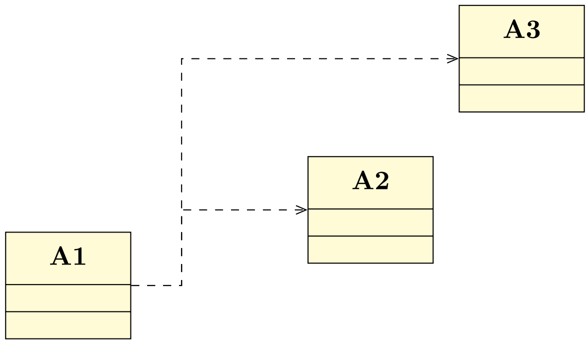


In some cases, this option is not very convenient, because it needs to compute the option value to give. There is another possibility by uusing the `arm1` and `arm2` options, that control the size of the first and last segment respectively. Let's see the 2 following examples using respectively the `weight` option and the `arm1` option :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=1]{A2}
\umlemptyclass[x=6, y=3]{A3}

\umlHVHdep[weight=0.375]{A1}{A2}
\umlHVHdep[weight=0.25]{A1}{A3}
\end{tikzpicture}
```

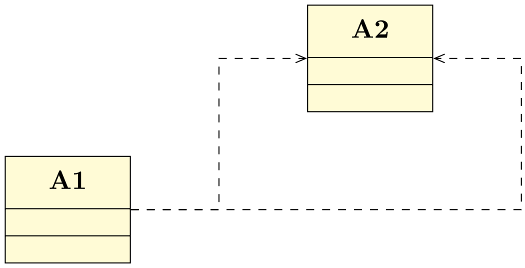



```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=1]{A2}
\umlemptyclass[x=6, y=3]{A3}

\umlHVHdep[arm1=1.5cm]{A1}{A2}
\umlHVHdep[arm1=1.5cm]{A1}{A3}
\end{tikzpicture}
```



The `arm1` and `arm2` options also take negative values. How does it work then ? A positive value means an arm oriented to the right direction (to the right or to the top), whereas a negative valuemeans an arm oriented to the opposite direction, that allows you to draw other 3-segments relations :

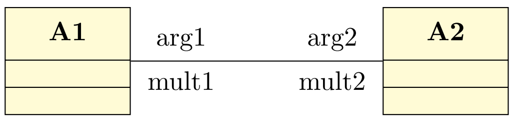
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=2]{A2}

\umlHVHdep[arm2=-2cm]{A1}{A2}
\umlHVHdep[arm2=2cm]{A1}{A2}
\end{tikzpicture}
```



1.2.4 To define informations about attributes of a relation

A relation means a dependency between two classes and represents an attribute in most of the cases. You can define its name with the `arg1` option or the `arg2` option, and its multiplicity with the `mult1` option or the `mult2` option :

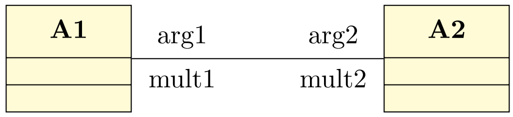
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=5]{A2}
\umlassoc[arg1=arg1, mult1=mult1, arg2=
    arg2, mult2=mult2]{A1}{A2}
\end{tikzpicture}
```



For unidirectional relations, you should use only `arg2` and `mult2` options. That is why shortcuts have been defined, namely the `arg` option and the `mult` option respectively.

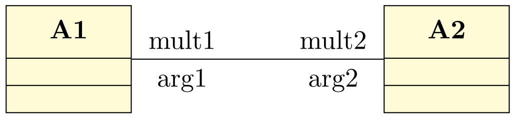
In addition, when you define the name and the multiplicity of an attribute, you may prefer use the all-in-one following options `attr1`, `attr2` and `attr` :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=5]{A2}
\umlassoc[attr1=arg1|mult1, attr2=arg2|
    mult2]{A1}{A2}
\end{tikzpicture}
```



This has an advantage : the semantic of the two values has disappeared and you can switch them for convenience :

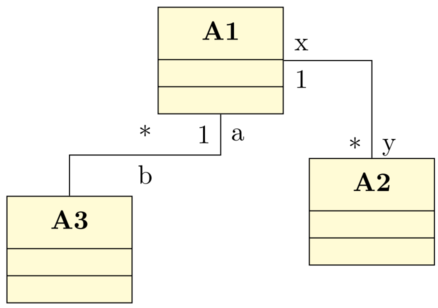
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=5]{A2}
\umlassoc[attr1=mult1|arg1, attr2=mult2|
    arg2]{A1}{A2}
\end{tikzpicture}
```



1.2.5 To place information about attributes of a relation

You can place information seen in previous section with the following options : `pos1`, `pos2` and `pos`. The `umlrelation` command determine by itself if name and multiplicity should be written on left and right or on top and bottom of the arrow, according to the geometry and their placement. For those who know TikZ enough, the mechanism is based on `auto` and `swap` options.

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=2,y=-2]{A2}
\umlemptyclass[x=-2,y=-2.5]{A3}
\umlassoc[geometry=-|,arg1=x,mult1=1,pos
    1=0.2,arg2=y,mult2=*,pos2=1.9]{A1}{A2}
\umlassoc[geometry=|-|,arg1=a,mult1=1,pos
    1=0.5,arg2=b,mult2=*,pos2=1.5]{A1}{A3}
\end{tikzpicture}
```

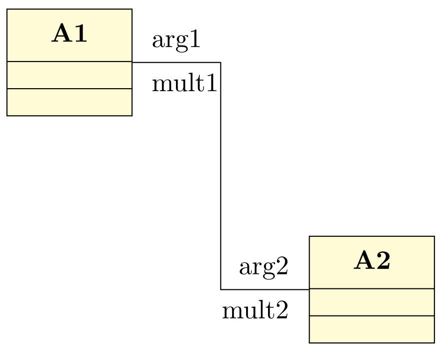


You may have noticed that the range of values of the position depends on the number of segments composing the arrow. For a straight line, position has to be between 0 (source class) and 1 (target class). If there are 2 segments, then position has to be between 0 et 2 (target class), the value 1 corresponding to the control node. Otherwise, position has to be between 0 et 3, values 1 and 2 corresponding to the first and second control node respectively.

1.2.6 To adjust the alignment of information about attributes of a relation

Name and multiplicity of an attribute, when they are written on top and bottom of the relation, are centered by default. You can define an other alignment. The options `align1`, `align2` and `align` are used to have ragged right or ragged left text :

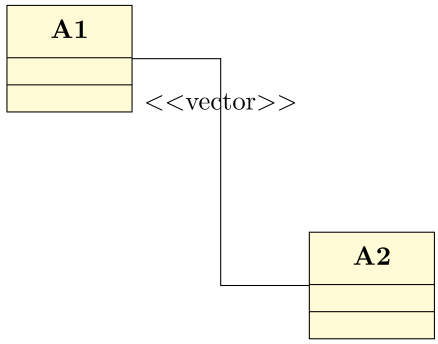
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=-3]{A2}
\umlassoc[geometry=-|-, arg1=arg1, mult1=
    mult1, pos1=0.1, align1=left , arg2=arg
    2, mult2=mult2, pos2=2.9, align2=right
    ]{A1}{A2}
\end{tikzpicture}
```



1.2.7 To define and place the stereotype of a relation

The stereotype of a relation is a keyword contained between ≪ and ≫. You can define it with the option `stereo` and place it with the option `pos stereo`.

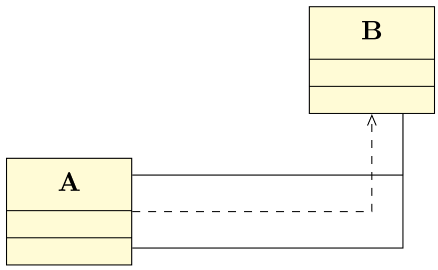
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=-3]{A2}
\umlassoc[geometry=-|-,stereo=vector, pos
    stereo=1.2]{A1}{A2}
\end{tikzpicture}
```



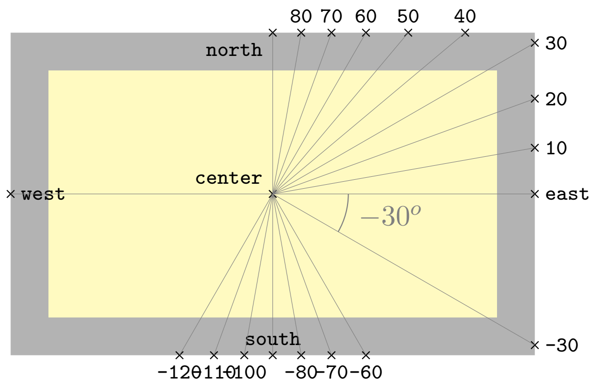
1.2.8 To modify the anchor points of a relation

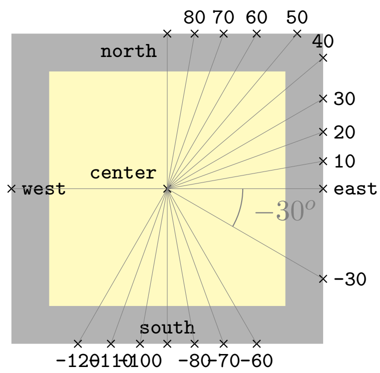
The default behavior of a relation is to start from the center anchor of the source class node and to end to the center anchor of the target class node. You can adjust this with the options `anchor1` and `anchor2`.

```
\begin{tikzpicture}
\umlemptyclass{A}
\umlemptyclass[x=4,y=2]{B}
\umldep[geometry=-|]{A}{B}
\umlassoc[geometry=-|, anchor1=30, anchor2=300,
   name=assoc1]{A}{B}
\umlassoc[geometry=-|, anchor1=-30, anchor2=-60,
   name=assoc2]{A}{B}
\end{tikzpicture}
```



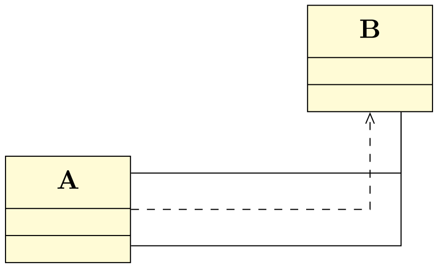
You give angular values in degree and they can be negative. The internal mechanism of TikZ uses modulos. The value 0 is east, 90 is north, 180 (or -180) is west, et 270 (or -90) is south. The following figure illustrates this option and its angular meaning on 2 examples of rectangular nodes, (class nodes for instance). You can notice that border anchors (to use TikZ vocabulary) depend on node dimensions.





You will very often define `anchor1` and `anchor2` simultaneously. In this case, you can use the all-in-one option `anchors` :

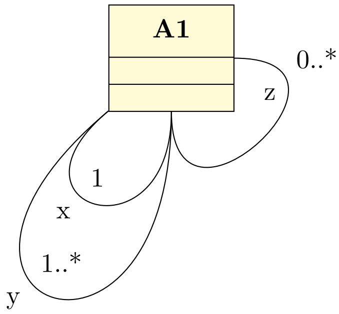
```
\begin{tikzpicture}
\umlemptyclass{A}
\umlemptyclass[x=4,y=2]{B}
\umldep[geometry=-|]{A}{B}
\umlassoc[geometry=-|, anchors=30 and 300, name=
    assoc1]{A}{B}
\umlassoc[geometry=-|, anchors=-30 and -60, name=
    assoc2]{A}{B}
\end{tikzpicture}
```



1.2.9 To define a recursive relation

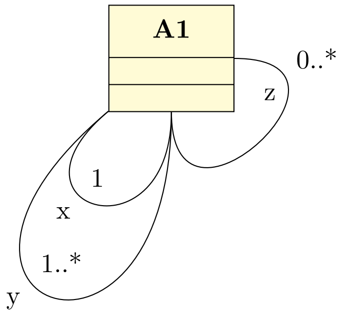
You can define recursive relations, namely a relation from a class to itself. Then, the `geometry` option is disabled, but 3 specific options are available : `angle1` determines the start angle, `angle2` determines the end angle, and `loopsize` controls the size of the loop.

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlassoc[arg=x,mult=1,pos=0.6, angle
   1=-90, angle2=-140, loopsize=2cm]{A1}{
   A1}
\umlassoc[arg=y,mult=1..*,pos=0.6, angle
   1=-90, angle2=-140, loopsize=4cm]{A1}{
   A1}
\umlassoc[arg=z,mult=0..*,pos=0.8, angle
   1=-90, angle2=0, loopsize=2cm]{A1}{A1}
\end{tikzpicture}
```



When you use recursive relations, you will notice that you will need the 3 options simultaneously. This is the reason why a compact form is defined, the `recursive` option, and the following syntax :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlassoc[arg=x, mult=1, pos=0.6,
    recursive=-90|-140|2cm]{A1}{A1}
\umlassoc[arg=y, mult=1..*, pos=0.6,
    recursive=-90|-140|4cm]{A1}{A1}
\umlassoc[arg=z, mult=0..*, pos=0.8,
    recursive=-90|0|2cm]{A1}{A1}
\end{tikzpicture}
```



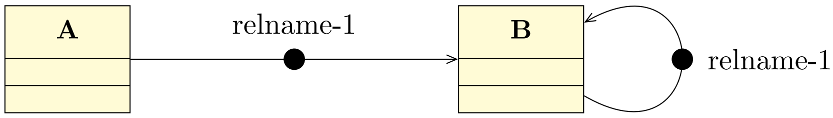
1.2.10 Name of auto-built points of a relation

In order to understand the purpose of giving a name to a relation, one should explain how arrows are defined.

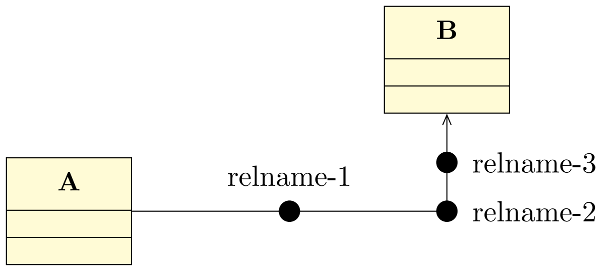
To build an arrow, we need to define control nodes, and a name for each one. The only way to identify a relation is to give a name using a id counter. This counter is incremented each time we define a relation in a picture. Let's suppose the relation has the id i. The name of the relation, called relname in the following, is : relation-i

The first defined node is the middle of the class nodes. It is called *relname-middle*. To simplify, we will not deal with the placement of the argument and its multiplicity here. So, there are 3 cases :

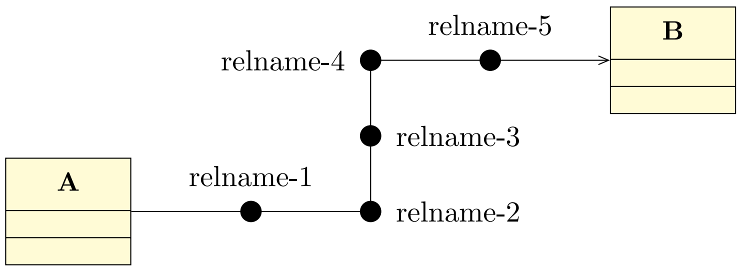
1. If the arrow is a straight line or a recursive line, it is renamed in *relname-1*.



2. If the arrow has one right angle, the node placed at the angle is named *relname-2*, that is enough to draw the arrow. 2 other nodes are defined, placed at the middle of each arc and named respectively *relname-1* and *relname-3*.



3. If the arrow has 2 right angles, they are defined with *relname-middle*, that is enough to draw the arrow. Nodes placed at the angles are named respectively *relname-2* and *relname-4*. 3 other nodes are defined, at the middle of each arc, named respectively *relname-1*, *relname-3*, and *relname-5*.

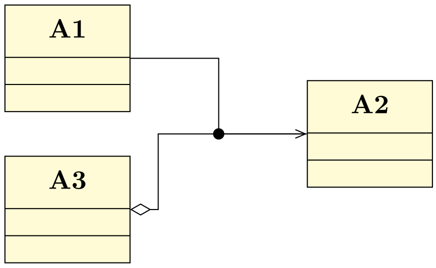


This default behavior is not easy to use, because the value of the counter is not defined by the user, and depends on the order of definition of the relations in the picture. This is the reason why you can define *relname* thanks to the `name` option. In the two following sections, you will see when this option is useful.

1.2.11 To draw an intersection point between relations

When you draw a diagram, it occurs that relations cross other ones or share arcs. Let's take two crossing arrows. Can both start points go graphically to both end points ? If yes, you will want to draw a point at the intersection of the arrows, and this point should be a control node of one the the relations. To define the point, you can use the `umlpoint` command.

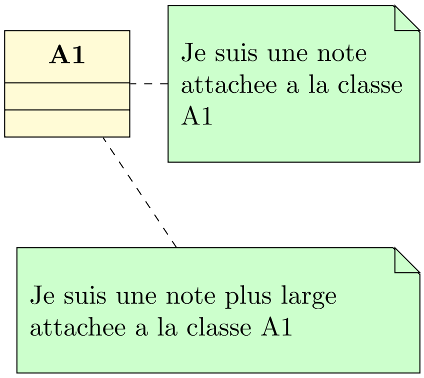
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=-1]{A2}
\umlemptyclass[y=-2]{A3}
\umlassoc[geometry=-|-, name=assoc]{A1}{A
   2}
\umluniaggreg[geometry=-|-, weight=0.3]{A
   3}{A2}
\umlpoint{assoc-4}
\end{tikzpicture}
```



1.3 Comments / constraints note

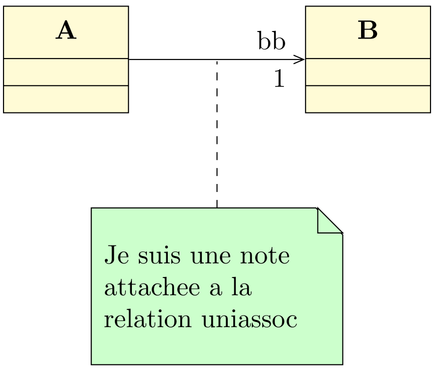
A note is a text comment attached to a class or a relation. The `umlnote` command needs the name of the node as argument :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlnote[x=3]{A1}{Je suis une note
   attachee a la classe A1}
\umlnote[x=2,y=-3, width=5cm]{A1}{Je suis
    une note plus large attachee a la
   classe A1}
\end{tikzpicture}
```



Here again, you can give the name of a control node of a relation to attach the note. Giving a name to the relation will be very useful :

```
\begin{tikzpicture}
\umlemptyclass{A}
\umlemptyclass[x=4]{B}
\umluniassoc[arg=bb, mult=1, pos=0.95, align=
   right, name=uniassoc]{A}{B}
\umlnote[x=2,y=-3]{uniassoc-1}{Je suis une note
   attachee a la relation uniassoc}
\end{tikzpicture}
```



Notes have 2 uses : comments and contraints (generally in OCL format).

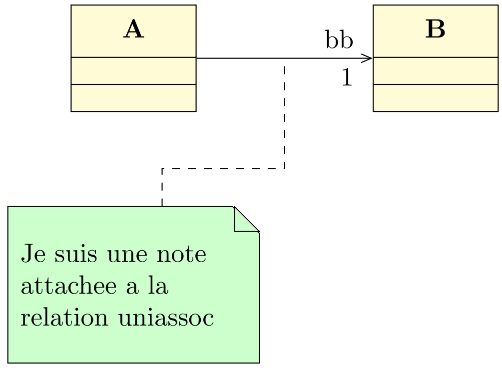
The `umlnote` command has the following options :

x, y These 2 options define the coordinates of the note.

width This option defines the width of the note. For TikZ users, it encapsulates the `text width` option

weight, arm, anchor1, anchor2, anchors These options has the same behavior as for `umlrelation`, `arm` being equivalent to `arm1`, namely attached to the note.

```
\begin{tikzpicture}
\umlemptyclass{A}
\umlemptyclass[x=4]{B}
\umluniassoc[arg=bb, mult=1, pos=0.95, align=
   right , name=uniassoc]{A}{B}
\umlnote[y=-3, geometry=|-|, anchor1=70, arm=0.5
   cm]{uniassoc-1}{Je suis une note attachee a la
     relation uniassoc}
\end{tikzpicture}
```



For a note, you can also use the `geometry` option, as for `umlrelation`. Then, aliases have been defined : `umlHVnote`, `umlVHnote`, `umlHVHnote` and `umlVHVnote`.

⚠ For each of these aliases, the `geometry` option is forbidden.

⚠ There is no alias for the - - value. It is the default one.

1.4 To change preferences

Thanks to the `tikzumlset` command, you can change default preferences for packages, classes and notes. The available options are :

text : allows you to set text color for every drawn object (=black by default),

draw : allows you to define edge color for every drawn object (=black by default),

fill class : allows you to define the background color of a class node (=yellow!20 by default),

fill template : allows you to define the background color of a template node (=yellow!2 by default),

fill package : allows you to define the background color of a package (=blue!20 by default),

fill note : allows you to define the background color for a note (=green!20 by default),

font : allows you to define the font style for every drawn object (=\small by default).

Furthermore, relation commands has the `style` option taking a TikZ style name as value.

Let's see the definition of the `umlinherit` command :

```
\tikzstyle{tikzuml inherit style}=[color=\tikzumldrawcolor, -open triangle 45]
\newcommand{\umlinherit}[3][]{\umlrelation[style={tikzuml inherit style},
   #1]{#2}{#3}}
```

You can easily define a command on this model by defining a particular style.

1.5 Examples

1.5.1 Example from introduction, step by step

We will now build step by step the picture seen in introduction to understand the behavior of each used command.

Definition of packages p, sp1 and sp2

The package p is placed at (0,0) (default), and the sub-packages sp1 and sp2 respectively at (0,0) and (10,-6).

```
\begin{tikzpicture}
\begin{umlpackage}{p}
  \umlvirt{setB(b : B) : void} \\ getB() : B}
\end{umlpackage}
}{}
  }{}
```

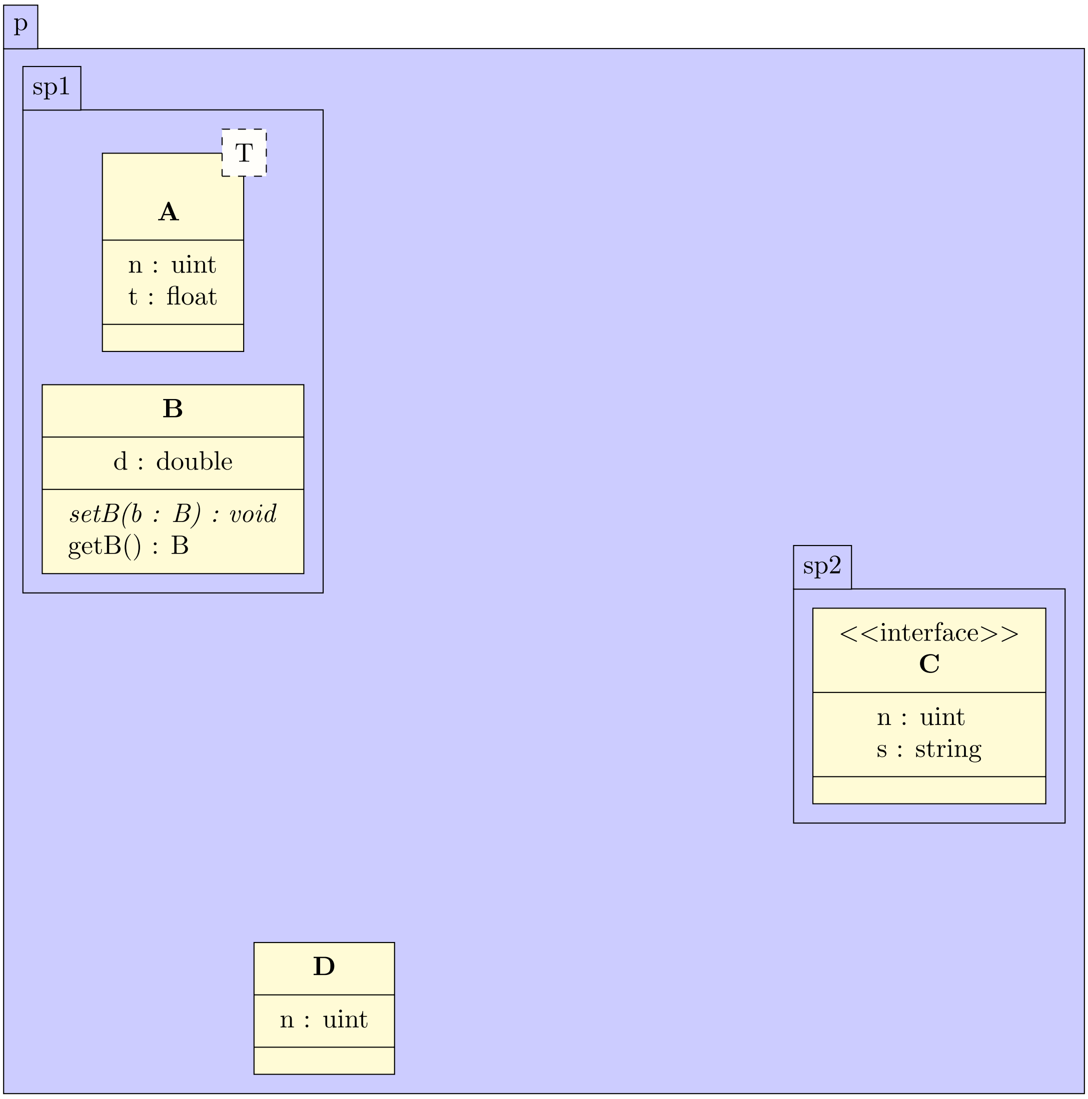
Definition of classes A, B, C, D and their attributes and operations

The class A is placed at (0,0) in the sub-package sp1 and has a template parameter : T. The class B is placed 3 units below A, still in the sub-package sp1. The interface C is placed at (0,0) in the sub sp2. The class D is placed at (2,-11) in the package p.

Class A has two attributes. Class B has one attribute and two operations (one is virtual). Class C has two attributes. Classe D has one attribute.

```
\begin{tikzpicture}
\begin{umlpackage}{p}
\begin{umlpackage}{sp1}
\umlclass[template=T]{A}{
  n : uint \\ t : float
}{}
\umlclass[y=-3]{B}{
  d : double
}{
  \umlvirt{setB(b : B) : void} \\ getB() : B}
\end{umlpackage}
\begin{umlpackage}[x=10,y=-6]{sp2}
\umlinterface{C}{
  n : uint \\ s : string
}{}
```

```
\end{umlpackage}
\umlclass[x=2,y=-10]{D}{
  n : uint
  }{}
```

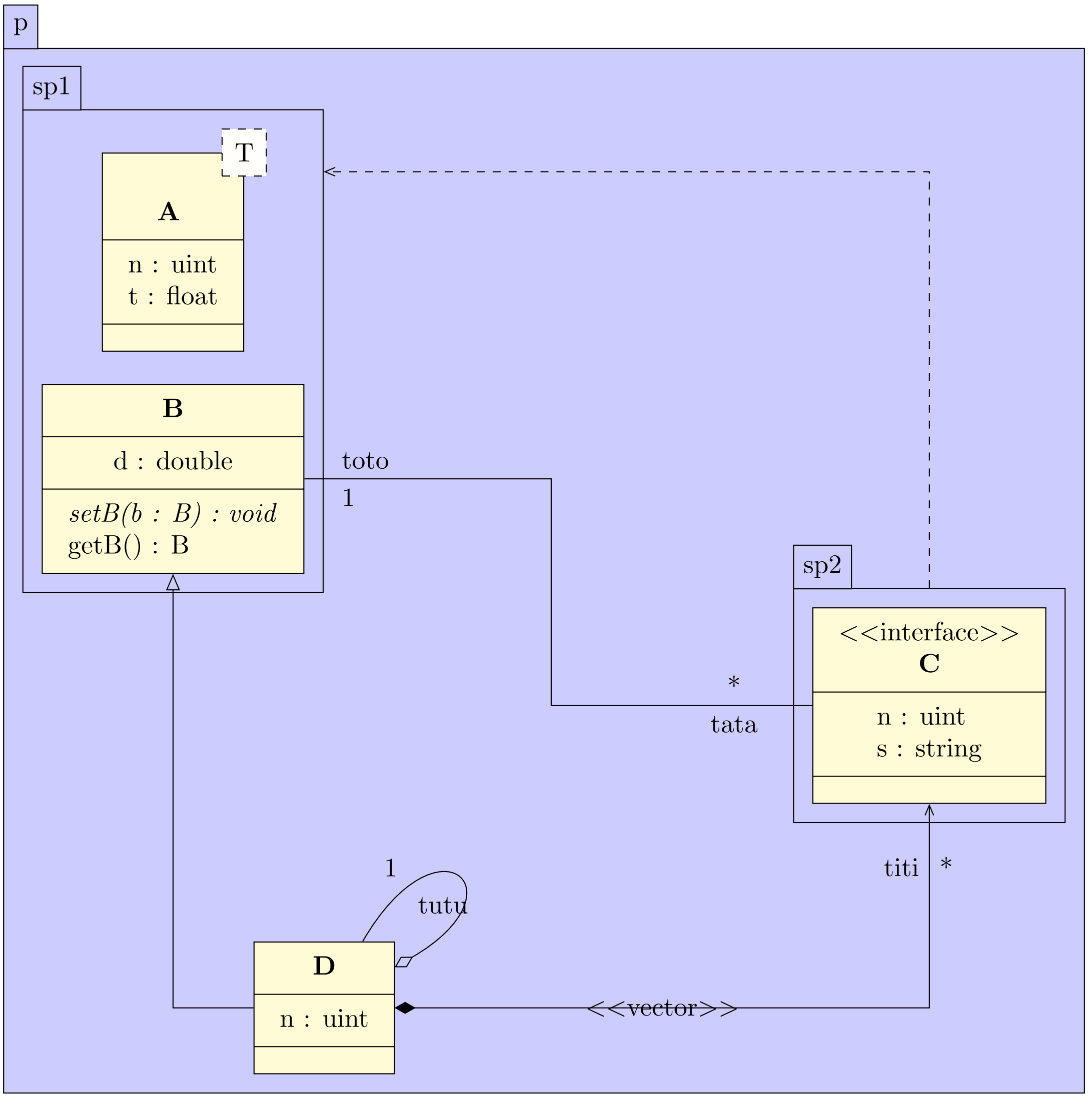


Definition of relations

We define an association between classes C and B, a unidirectional composition between classes D and C, an import relation named "import" between sub-packages sp2 and sp1 (with modification of anchors), a recursive aggregation on class D and an inheritnce between classes D and B. On thses relations, we will specify argument names and multiplicities. You can notice the value given to place these elements on each arrow according to the geometry.

...

```
\end{umlpackage}

\umlassoc[geometry=-|-, arg1=tata, mult1=*, pos1=0.3, arg2=toto, mult2=1, pos
    2=2.9, align2=left]{C}{B}
\umlunicompo[geometry=-|, arg=titi, mult=*, pos=1.7, stereo=vector]{D}{C}
\umlimport[geometry=|-, anchors=90 and 50, name=import]{sp2}{sp1}
```

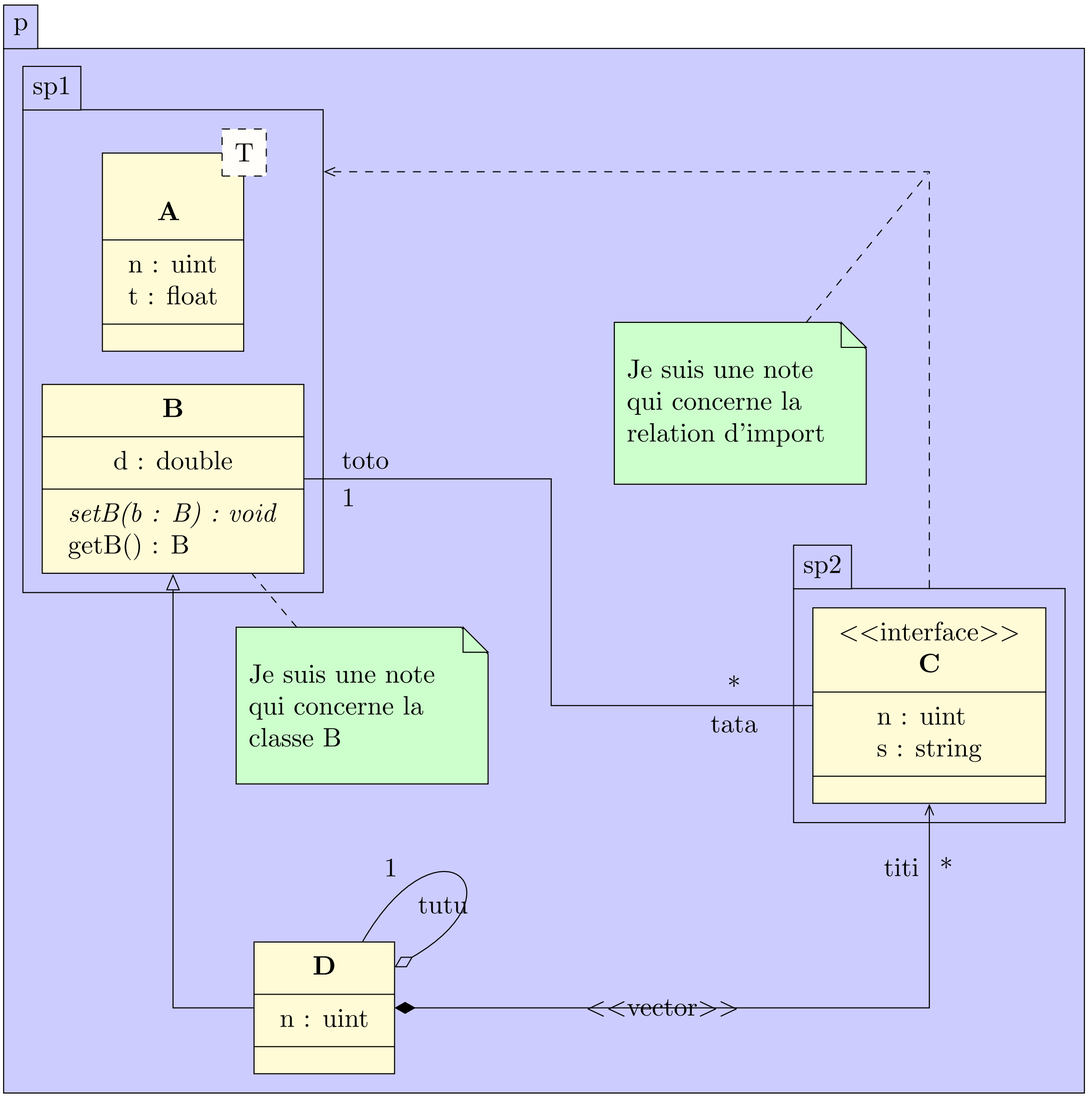


Definition of notes

We add a note attached to class B and a note attached to the import relation.

...

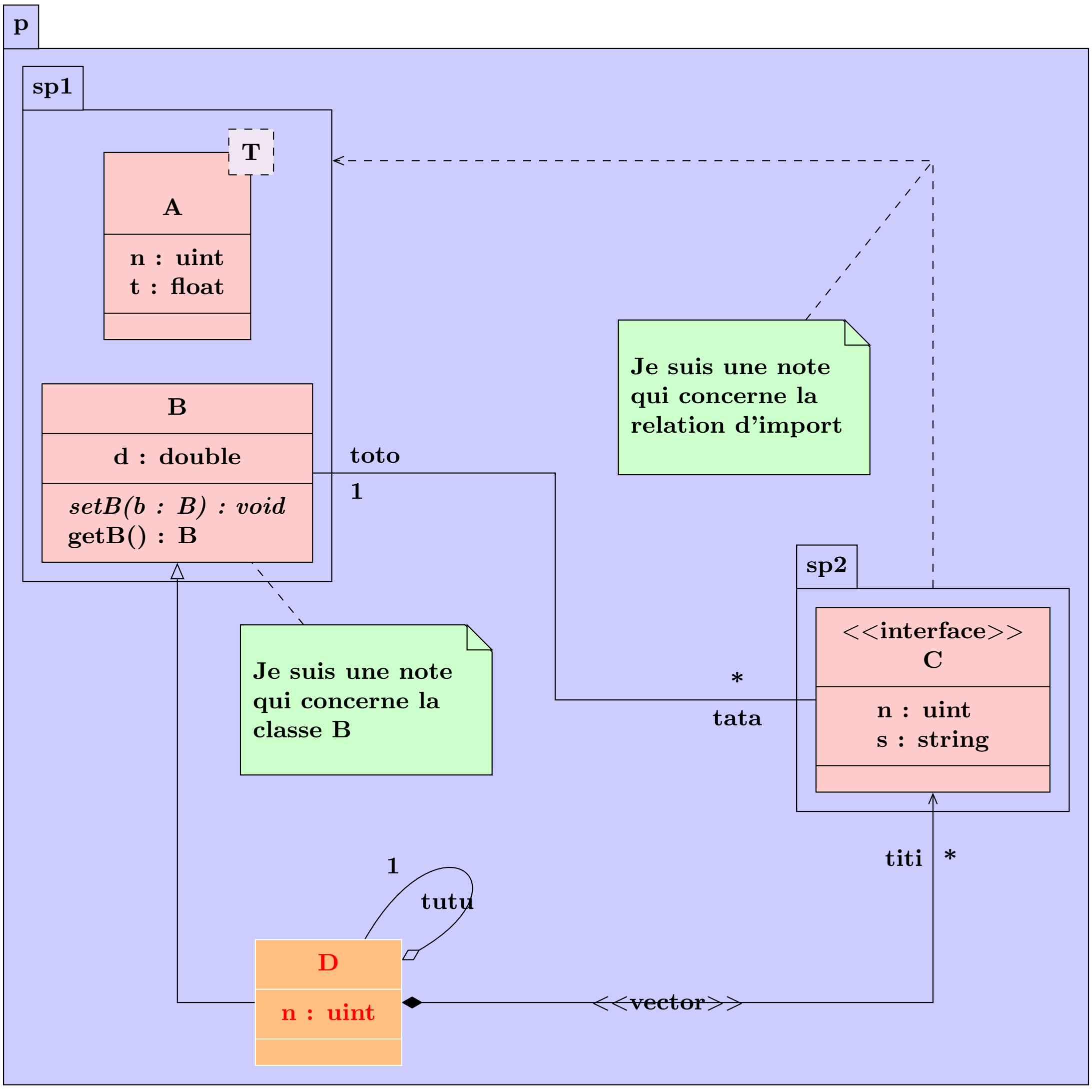
```
\umlaggreg[arg=tutu, mult=1, pos=0.8, angle1=30, angle2=60, loopsize=2cm]{D}{D}
\umlinherit[geometry=-|]{D}{B}
\umlnote[x=2.5,y=-6, width=3cm]{B}{Je suis une note qui concerne la classe B}
\umlnote[x=7.5,y=-2]{import-2}{Je suis une note qui concerne la relation d'import}
\end{tikzpicture}
```



Setting style

We illustrate the use of the `tikzumlset` command by changing colors associated to class and font. We can also change colors of a given class with `draw`, `text` and `fill` options.

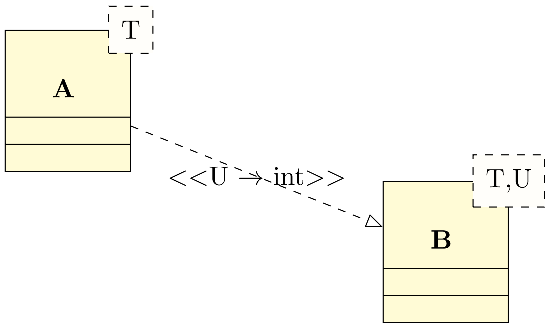
```
\tikzumlset{fill class=red!20, fill template=violet!10, font=\bfseries\
  footnotesize}
\begin{tikzpicture}
...
\umlclass[x=2,y=-11, fill=orange!50, draw=white, text=red]{D}{
  n : uint
  }{}
...
\end{tikzpicture}
```



1.5.2 To define a specialization of a class

A specialization of a classe is an inheritance from a templace class in which one of the template parameters is defined. To draw this relation, you will use the `umlreal` command , and its `stereo` option :

```
\begin{tikzpicture}
\umlemptyclass[template=T]{A}
\umlemptyclass[template={T,U}, x=5, y
    =-2]{B}
\umlreal[stereo={U $\rightarrow$ int}]{A
    }{B}
\end{tikzpicture}
```

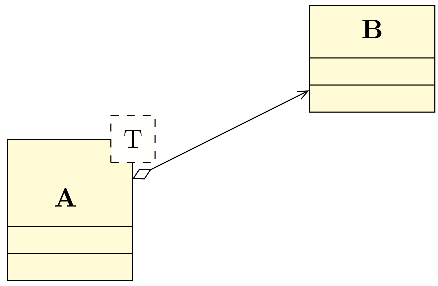


1.6 Priority rules of options and known bugs

1. The `geometry` option has always the priority on the others options. It means for instance that if it has a non-default value, then `angle1`, `angle2` and `loopsize` options, defining recursive relations, will be ignored.

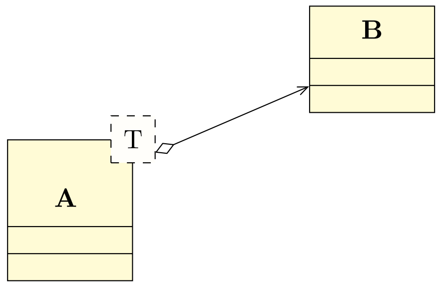
2. As far as a template class is concerned, there are cases in which a relation about it will not be drawn correctly, as in the picture below, where the aggregation symbol is hidden by the template parameter :

```
\begin{tikzpicture}
\umlemptyclass[template=T]{A}
\umlemptyclass[x=4,y=2]{B}
\umluniaggreg{A}{B}
\end{tikzpicture}
```



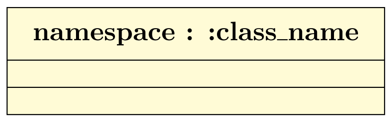
To solve this problem, you van link the arrow between the template part of class A and class B, by adding the suffix -template and adjusting the start anchor (the -30 value is correct here) :

```
\begin{tikzpicture}
\umlemptyclass[template=T]{A}
\umlemptyclass[x=4,y=2]{B}
\umluniaggreg[anchor1=-30]{A-template}{B}
\end{tikzpicture}
```



3. If you define a class with a name having the : character in it – typically when you give the namespace of the class – it may have a conflict with the french (or frenchb or francais) option of the babel package. Indeed, these options add a white-space before the : character if the writer forgot it, that is a problem for the access operator ::. If we take the example of class definition, we should obtain :

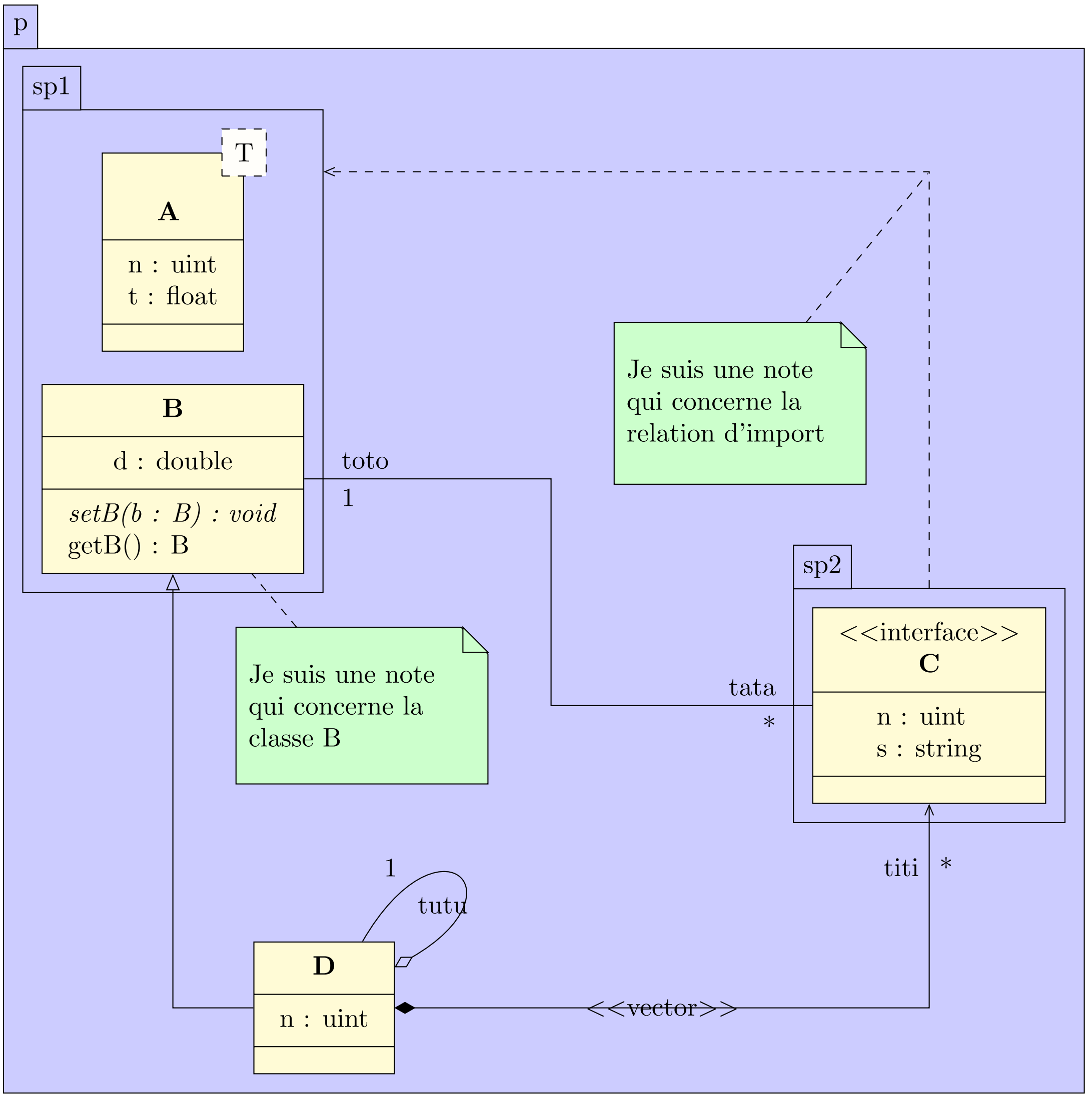
```
\begin{tikzpicture}
\umlclass[x=0,y=0]{namespace::class\_name}{}{}
\end{tikzpicture}
```



The solution is to use a specific macro given by these options of babel package you have to use in the preamble of your document :

```
\frenchbsetup{AutoSpacePunctuation=false}
```

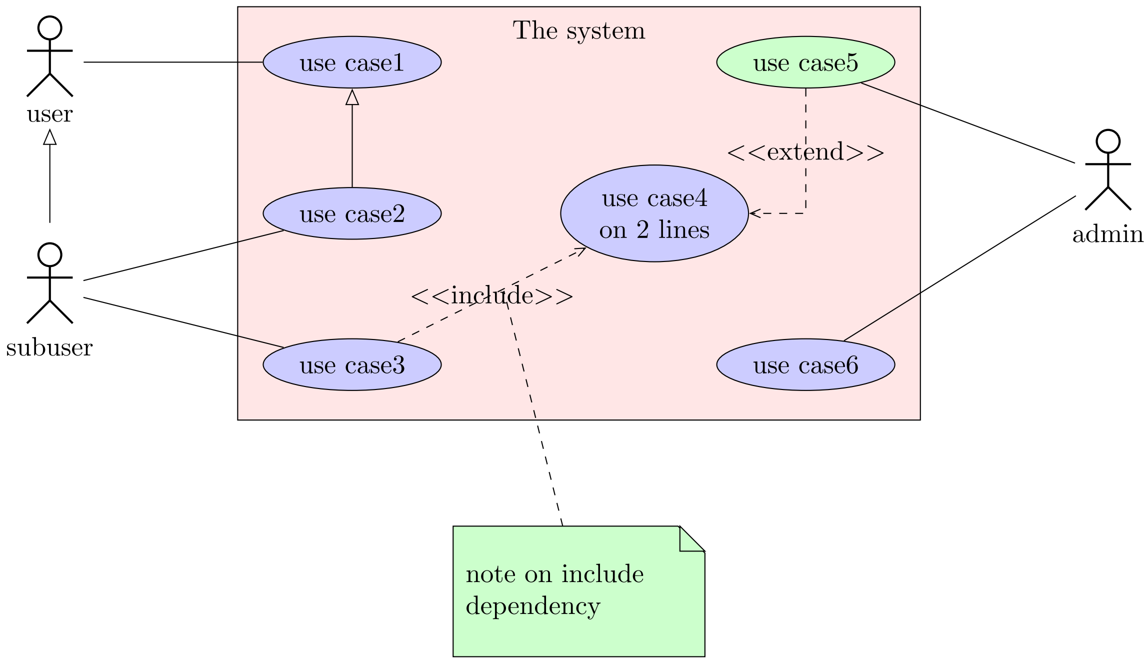
4. The automatic placement of argument names and multiplicity on a relation can be surprising when you can to deactivate it. Let's take the example shown in introduction. If we focus on the association relation and its attributes *toto* and *tata*, *toto* is above, *tata* is below. If we justify to the right the *tata* attribute (and change its position to 0.1), positions of *tata* and its multiplicity exchange.


p
sp1
T
A
n : uint
t : float
B
d : double
setB(b : B) : void
getB() : B
toto
1
Je suis une note
qui concerne la
relation d'import
sp2
<<interface>>
C
n : uint
s : string
tata
*
Je suis une note
qui concerne la
classe B
1
tutu
titi
*
D
n : uint
<<vector>>

Chapter 2

Use case diagrams

Here is an example of use case diagram you can draw :

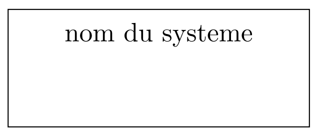


We will see how to define the four constitutive elements of such a diagram : the system, the actors, the use cases and the relations.

2.1 To define a system

A system is defined by the `umlsystem` environment :

```
\begin{tikzpicture}
\begin{umlsystem}[x=0, y=0]{nom du systeme}

\end{umlsystem}
\end{tikzpicture}
```

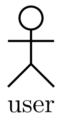


Both options `x` and `y` allow to place the system in the picture. The default value is 0. Inside this environment, you will define use cases, whereas outside, you will define actors.

2.2 To define an actor

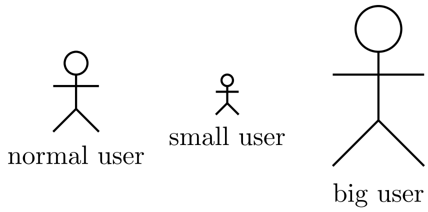
You can define an actor with the `umlactor` command :

```
\begin{tikzpicture}
\umlactor[x=0, y=0]{user}
\end{tikzpicture}
```



Both options `x` and `y` allow to place the actor in the picture. The default value is 0. You can change dimensions of the actor symbol with the `scal` option. It also adapts position of the label below :

```
\begin{tikzpicture}
\umlactor{normal user}
\umlactor[x=2, scale=0.5]{small user}
\umlactor[x=4, scale=2]{big user}
\end{tikzpicture}
```



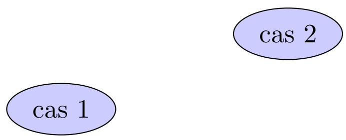
The actor symbol size is defined according to the font size (ex unit), whereas the distance between the symbol and the label is in cm. You can adjust it if you need with the `below` option (0.5cm by default).

```
\tikzumlset{font=\tiny}
\begin{tikzpicture}
\umlactor{normal user}
\umlactor[x=2, scale=0.5, below=0.1cm]{
   small user}
\umlactor[x=4, scale=2]{big user}
\end{tikzpicture}
```

2.3 To define a use case

You can define a use case with the `umlusecase` command :

```
\begin{tikzpicture}
\umlusecase[x=0, y=0]{cas 1}
\umlusecase[x=3, y=1]{cas 2}
\end{tikzpicture}
```



Both options `x` and `y` allow to place the use case in the picture or in the container system. The default value is 0. The text argument is the label of the use case. The node representing the use case has a default name, based on a global counter, that is like usecase-17. For pratical reasons, you can rename it thanks to the `name` option.

Furthermore, you can set the witdh of the use case with the `width` option.

Now, we can talk about relations between use cases, systems and actors.

2.4 To define a relation

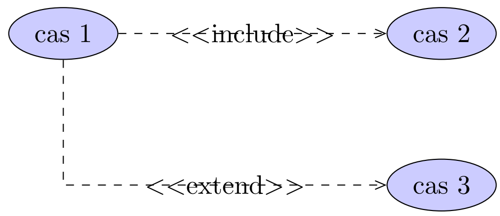
Relations in a user case diagram are of 4 categories :

- Inheritance relations, between actors or between use cases. You can use the `umlinherit` command and its aliases, ie subsection 1.2.1.
- Association relations, between an actor and a use cases. You can use the `umlassoc` command and its aliases, ie subsection 1.2.1.

- Include and extend relations. Graphically, it is a dependency relation, as for class diagrams, with the stereotype `extend` or `include`. You can use aliases of the `umlrelation` command, named `umlinclude`, `umlHVinclude`, . . . , `umlextend`, `umlHVextend`, . . . , to define such relations.

`anchor1`, `anchor2`, `anchors`, `arm1`, `arm2`, `weight`, `geometry` (only for `umlinclude` and `umlextend`), and `pos stereo` options are available here.

```
\begin{tikzpicture}
\umlusecase[name=case1]{cas 1}
\umlusecase[x=5, name=case2]{cas 2}
\umlusecase[x=5, y=-2, name=case3]{cas 3}

\umlinclude{case1}{case2}
\umlVHextend[pos stereo=1.5]{case1}{case3}
\end{tikzpicture}
```



2.5 To change preferences

With the `tikzumlset` command, you can change default colors for use cases, systems, actors and relations :

text : allows to set the text color (=black by default),

draw : allows to set the edge colors (=black by default),

fill usecase : allows to set the background color for use cases (=blue!20 by default),

fill system : allows to set the background color for systems (=white by default),

font : allows to set the font style (=\small by default).

You can also use `text`, `draw` and `fill` options on a particular element to change its colors, as shown in the introduction example.

2.6 Examples

2.6.1 Example from introduction, step by step

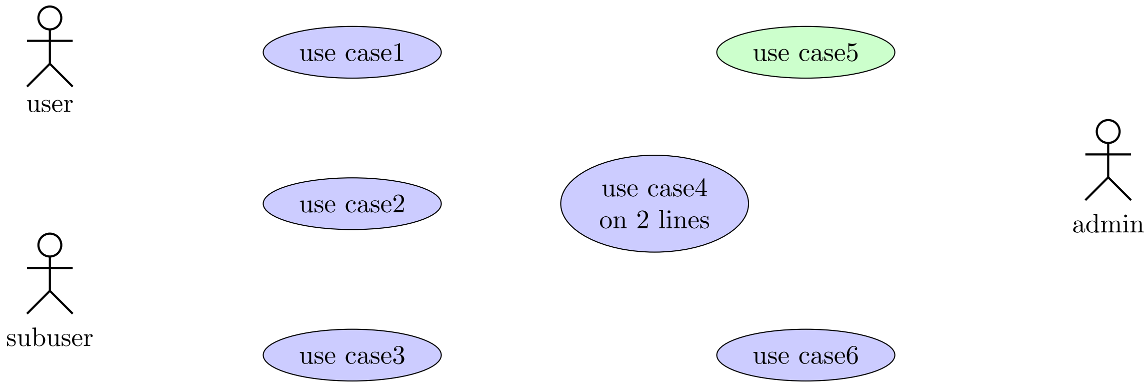
Definition of actors

```
\umlactor{user}
\umlactor[y=-3]{subuser}
\umlactor[x=14, y=-1.5]{admin}
```

Definition of use cases

We also show here the use of the `fil` option.

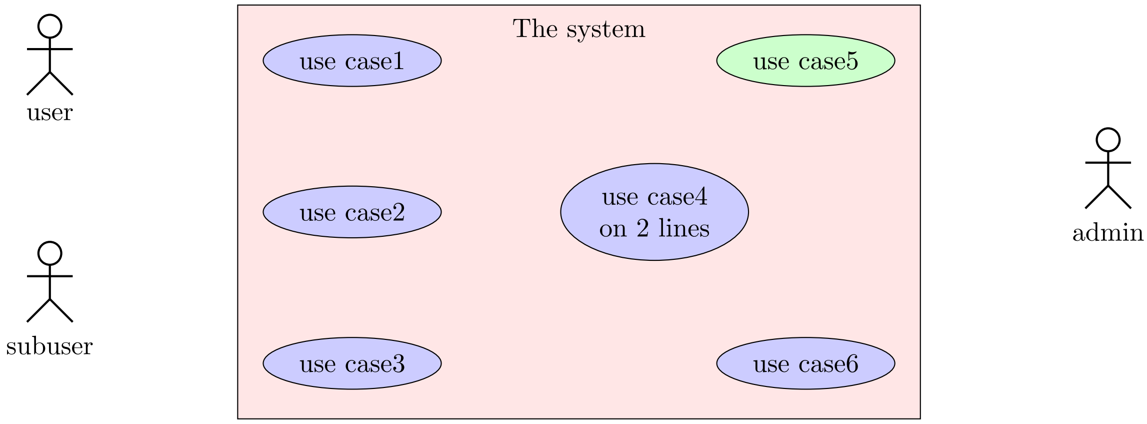
```
\umlusecase{use case1}
\umlusecase[y=-2]{use case2}
\umlusecase[y=-4]{use case3}
\umlusecase[x=4, y=-2, width=1.5cm]{use case4 on 2 lines}
\umlusecase[x=6, fill=green!20]{use case5}
\umlusecase[x=6, y=-4]{use case6}
\umlactor{user}
\umlactor[y=-3]{subuser}
\umlactor[x=14, y=-1.5]{admin}
```



Definition of the system

As the system is a box used as a new coordinate system, we have to change coordinates of use cases.

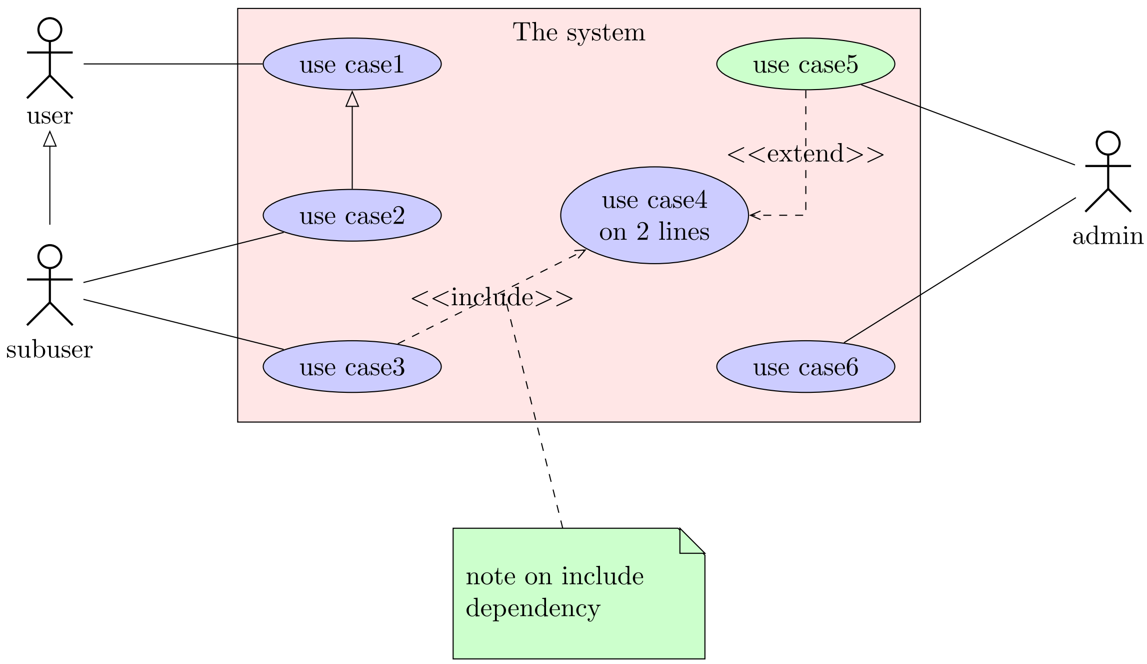
```
\begin{umlsystem}[x=4, fill=red!10]{The system}
\umlusecase{use case1}
\umlusecase[y=-2]{use case2}
\umlusecase[y=-4]{use case3}
\umlusecase[x=4, y=-2, width=1.5cm]{use case4 on 2 lines}
\umlusecase[x=6, fill=green!20]{use case5}
\umlusecase[x=6, y=-4]{use case6}
\end{umlsystem}

\umlactor{user}
\umlactor[y=-3]{subuser}
\umlactor[x=14, y=-1.5]{admin}
```



Definition of relations and of the note

You will notice here the use of the `name` option to ensure the definition of the note, and its interest for use cases, in order to ignore the order of their definition, as shown in the following example :

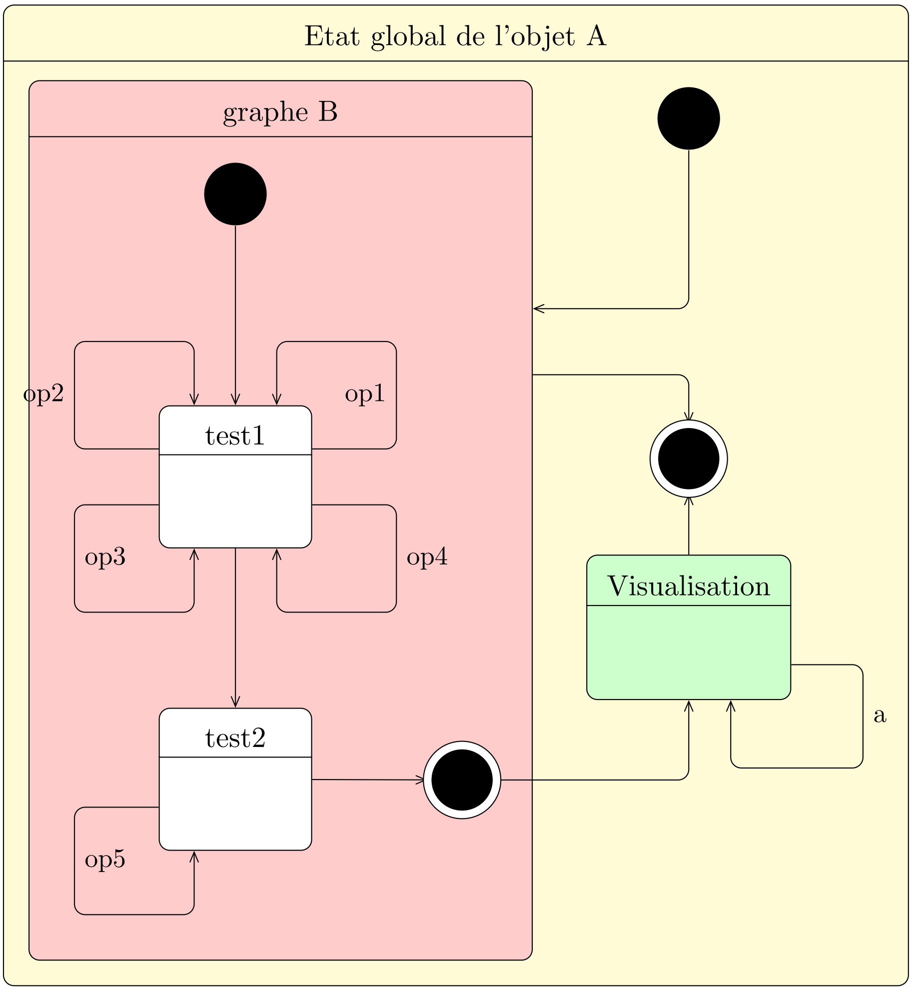
```
\begin{umlsystem}[x=4, fill=red!10]{The system}
\umlusecase{use case1}
\umlusecase[y=-2]{use case2}
\umlusecase[y=-4]{use case3}
\umlusecase[x=4, y=-2, width=1.5cm]{use case4 on 2 lines}
\umlusecase[x=6, fill=green!20]{use case5}
\umlusecase[x=6, y=-4]{use case6}
\end{umlsystem}

\umlactor{user}
\umlactor[y=-3]{subuser}
\umlactor[x=14, y=-1.5]{admin}

\umlinherit{subuser}{user}
\umlassoc{user}{usecase-1}
\umlassoc{subuser}{usecase-2}
\umlassoc{subuser}{usecase-3}
\umlassoc{admin}{usecase-5}
\umlassoc{admin}{usecase-6}
\umlinherit{usecase-2}{usecase-1}
\umlVHextend{usecase-5}{usecase-4}
\umlinclude[name=incl]{usecase-3}{usecase-4}

\umlnote[x=7, y=-7]{incl-1}{note on include dependency}
```



Chapter 3

State-transitions diagrams

Here is an example of state-transition diagram you can draw :

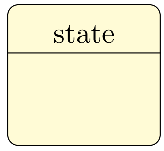


Now, we will see how to define parts of these diagrams, namely the ten sorts of state and the transitions.

3.1 To define a state

A "standard" state can be defined with the `umlstate` environment :

```
\begin{tikzpicture}
\begin{umlstate}[x=0, y=0, name=state]{
    state}

\end{umlstate}
\end{tikzpicture}
```

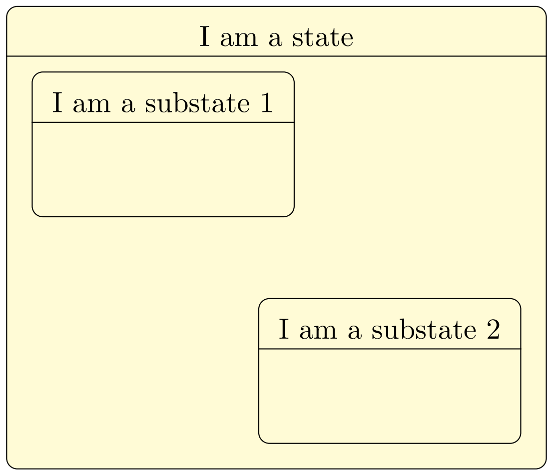


Both options `x` and `y` allows to place the state in the figure, or in another state. The default value is 0. The argument to give is the label of the state. The node representing the state has a default name, based on a global counter. For practical reasons, when you define a transition for instance, you can rename it with the `name` option.

You can also define the maximal width of a state with the `width` (10ex by default).

You can define a state in another state. Then, the coordinates of the sub-states are relative to the parent state :

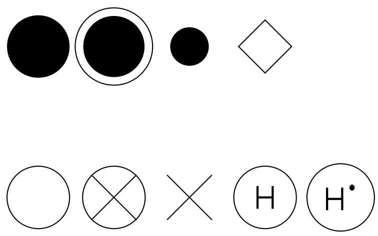
```
\begin{tikzpicture}
\begin{umlstate}[name=state]{I am a state}
\begin{umlstate}[name=substate1]{I am a
    substate 1}

\end{umlstate}
\begin{umlstate}[x=3, y=-3, name=substate
    2]{I am a substate 2}

\end{umlstate}
\end{umlstate}
\end{tikzpicture}
```



If you want to define a state without detailing it, you can use the `umlbasicstate` command, that is an alias of the `umlstate` environment.

Let's talk about the specific states :

```
\begin{tikzpicture}
\umlstateinitial[name=initial]
\umlstatefinal[x=1, name=final]
\umlstatejoin[x=2, name=join]
\umlstatedecision[x=3, name=decision]
\umlstateenter[y=-2, name=enter]
\umlstateexit[x=1, y=-2, name=exit]
\umlstateend[x=2, y=-2, name=end]
\umlstatehistory[x=3, y=-2, name=hist]
\umlstatedeephistory[x=4, y=-2, name=
    deephist]
\end{tikzpicture}
```



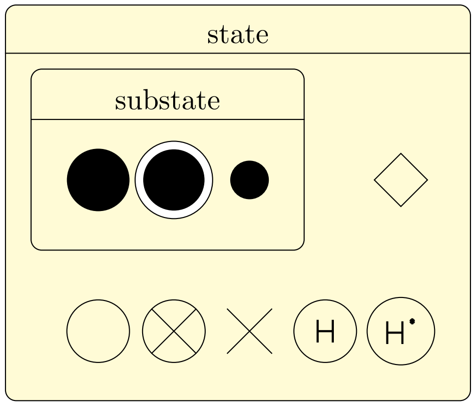
From left to right and top to bottom :

- An initial state is defined with the `umlstateinitial` command.
- A final state is defined with the `umlstatefinal` command.

- A join state is defined with the `umlstatejoin` command.
- A decision state is defined with the `umlstatedecision` command.
- An enter state is defined with the `umlstateenter` command.
- An exit state is defined with the `umlstateexit` command.
- An end state is defined with the `umlstateend` command.
- An history state is defined with the `umlstatehistory` command.
- A deep history state is defined with the `umlstatedeephistory` command.

These commands take several options : `name`, to rename the node, and `width` to set their size. You can use these commands in a `umlstate` environment :

```
\begin{tikzpicture}
\begin{umlstate}[name=state]{state}
\begin{umlstate}[name=substate]{substate}
\umlstateinitial[name=initial]
\umlstatefinal[x=1, name=final]
\umlstatejoin[x=2, name=join]
\end{umlstate}
\umlstatedecision[x=4, name=decision]
\umlstateenter[y=-2, name=enter]
\umlstateexit[x=1, y=-2, name=exit]
\umlstateend[x=2, y=-2, name=end]
\umlstatehistory[x=3, y=-2, name=hist]
\umlstatedeephistory[x=4, y=-2, name=
    deephist]
\end{umlstate}
\end{tikzpicture}
```



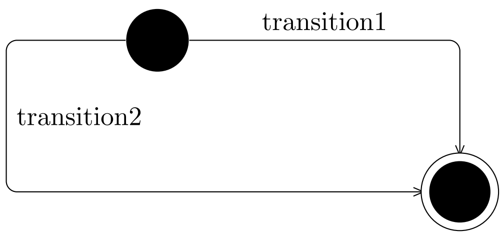
3.2 To define a transition

Transitions are relations between states in a state-transition diagram. You can define them with the `umltrans` command, that is an alias of the `umlrelation` command. There are unidirectional transitions and recursive transitions.

3.2.1 To define a unidirectional transition

Thanks to the `geometry` option, usual aliases are available : `umlHVtrans`, `umlVHtrans`, `umlVHVtrans` and `umlHVHtrans`. Graphically, the use of these aliases are the most interesting, because corners are rounded.

```
\begin{tikzpicture}
\umlstateinitial[name=initial]
\umlstatefinal[x=4, y=-2, name=final]
\umlHVtrans[arg=transition1, pos=0.5]{
    initial}{final}
\umlHVHtrans[arm1=-2cm, arg=transition2,
    pos=1.5]{initial}{final}
\end{tikzpicture}
```

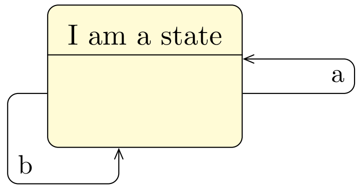


Every option of the `umlrelation` command can be used with the `umltrans` command and its aliases.

3.2.2 To define a recursive transition

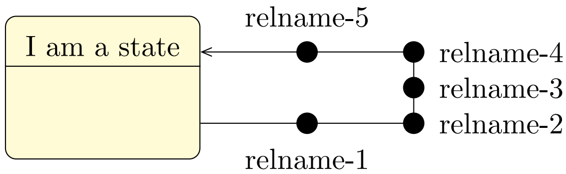
Recursive transitions are graphically the most difficult to manage, because their shape is a rounded rectangle, contrary to recursive relations in a class diagram. Conceptually, it is as if the `geometry` option has the value -|- or |-|, that is to say arrows composed of several segments.

```
\begin{tikzpicture}
\umlbasicstate[name=state]{I am a state}
\umltrans[recursive=-10|10|2cm, arg=a, pos
    =1.5, recursive direction=right to
    right]{state}{state}
\umltrans[recursive=-170|-110|2cm, arg=b,
    pos=2, recursive direction=left to
    bottom]{state}{state}
\end{tikzpicture}
```

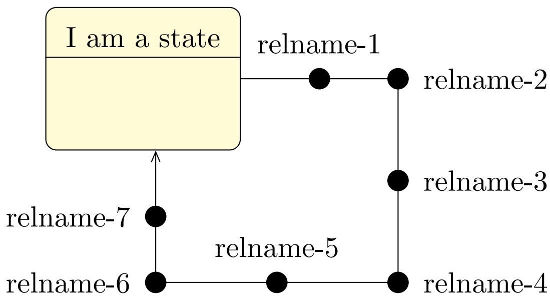


The `recursive direction` option is fundamental. Indeed, giving values of start angle and end angle is not enough to determine the start direction and the end direction of the recursive arrow, because it does not define the normal direction. Then, we have to precise it. There are 2 cases :

- The arrow can be composed of 3 segments. In this case, usable nodes are shown as follows :



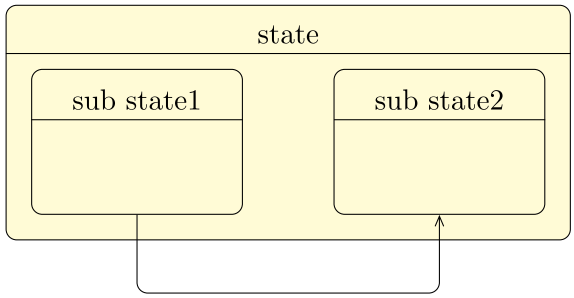
- The arrow can be composed of 4 segments. In this case, usable nodes are shown as follows :



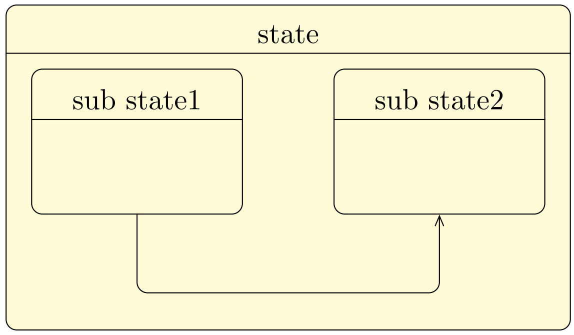
3.2.3 To define a transition between sub states

When you want to define transitions between sub-states, transitions are drawn inside the parent state.Then, you have to define them inside the `umlstate` environment. Let's compare the two following examples :

```
\begin{tikzpicture}
\begin{umlstate}[name=state]{state}
\umlbasicstate[name=substate1]{sub state1}
\umlbasicstate[x=4, name=substate2]{sub
    state2}
\end{umlstate}

\umlVHVtrans[arm1=-2cm]{substate1}{
    substate2}
\end{tikzpicture}
```




```
\begin{tikzpicture}
\begin{umlstate}[name=state]{state}
\umlbasicstate[name=substate1]{sub state1}
\umlbasicstate[x=4, name=substate2]{sub
    state2}

\umlVHVtrans[arm1=-2cm]{substate1}{
    substate2}
\end{umlstate}
\end{tikzpicture}
```



3.3 To change preferences

With the `tikzumlset` command, you can change default colors for states and transitions :

text : allows to set text color (=black by default),

draw : allows to set the edge color and the color of initial, final and join states (=black by default),

fill state : allows to set the background color of a state (=yellow!20 by default),

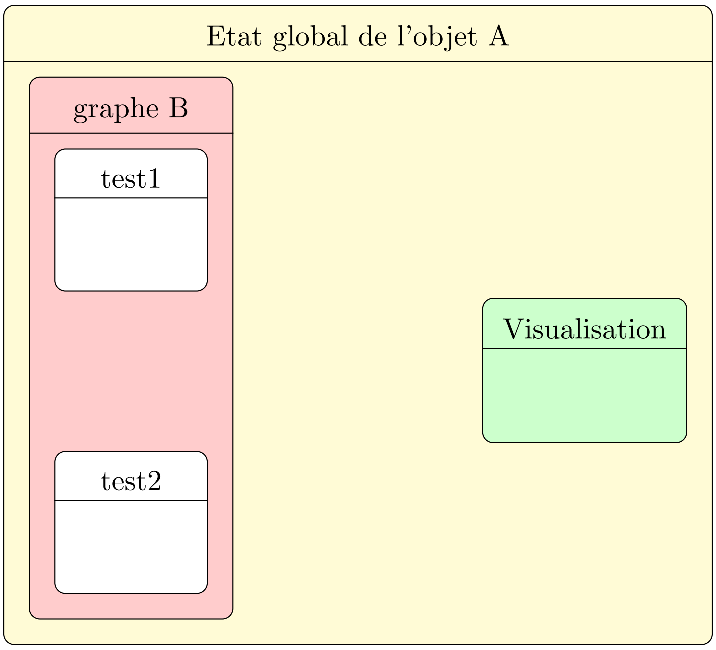
font : allows to set the font style (=\small by default).

You can also use the `text`, `draw` and `fill` options on a particular element, in order to change its colors, as shown in the introduction example.

3.4 Examples

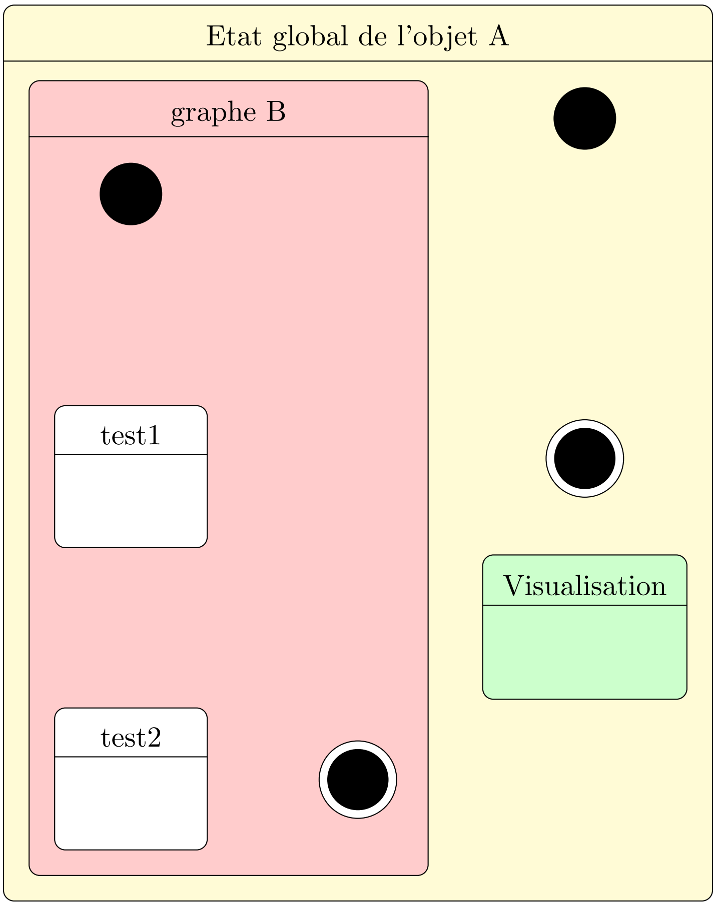
3.4.1 Example from introduction, step by step

Definition of basic states



```
\begin{umlstate}[name=Amain]{Etat global de l'objet A}
\begin{umlstate}[name=Bgraph, fill=red!20]{graphe B}
\umlbasicstate[y=-4, name=test1, fill=white]{test1}
\umlbasicstate[y=-8, name=test2, fill=white]{test2}
\end{umlstate}
\umlbasicstate[x=6, y=-6, name=visu, fill=green!20]{Visualisation}
\end{umlstate}
```

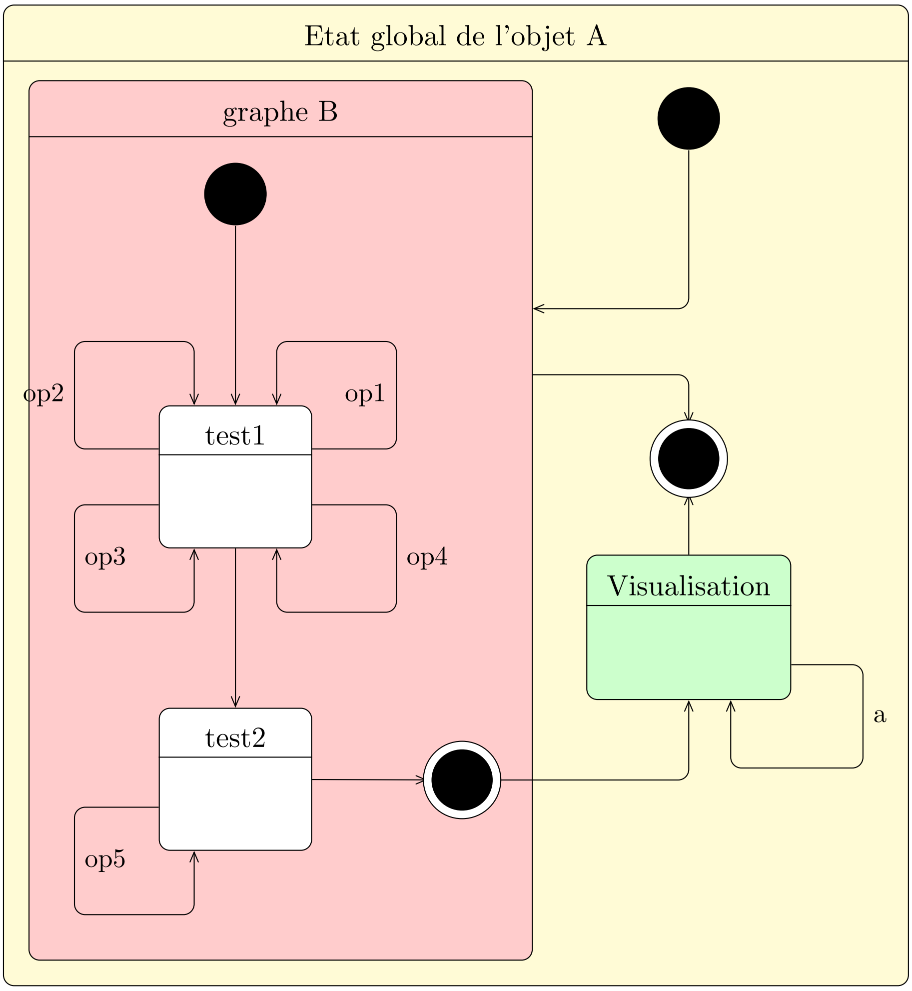
Definition of specific states




```
\begin{umlstate}[name=Amain]{Etat global de l'objet A}
\begin{umlstate}[name=Bgraph, fill=red!20]{graphe B}
\umlstateinitial[name=Binit]
\umlbasicstate[y=-4, name=test1, fill=white]{test1}
\umlbasicstate[y=-8, name=test2, fill=white]{test2}
\umlstatefinal[x=3, y=-7.75, name=Bfinal]
\end{umlstate}
\umlstateinitial[x=6, y=1, name=Ainit]
\umlstatefinal[x=6, y=-3.5, name=Afinal]
\umlbasicstate[x=6, y=-6, name=visu, fill=green!20]{Visualisation}
\end{umlstate}
```

Definition of transitions

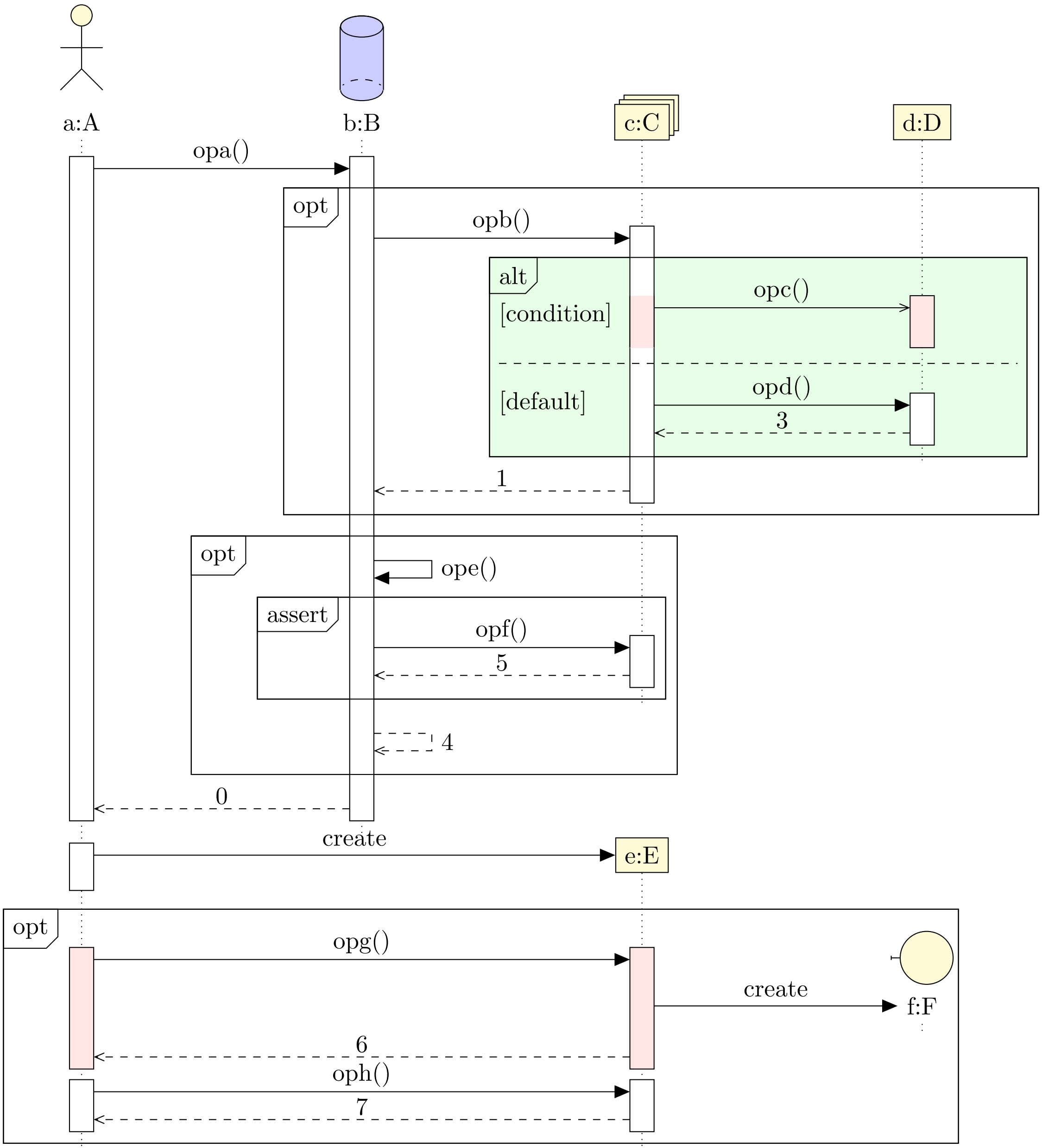
```
\begin{umlstate}[name=Amain]{Etat global de l'objet A}
\begin{umlstate}[name=Bgraph, fill=red!20]{graphe B}
\umlstateinitial[name=Binit]
\umlbasicstate[y=-4, name=test1, fill=white]{test1}
\umltrans{Binit}{test1}
\umltrans[recursive=20|60|2.5cm, recursive direction=right to top, arg={op1}, pos
   =1.5]{test1}{test1}
\umltrans[recursive=160|120|2.5cm, recursive direction=left to top, arg={op2}, pos
   =1.5]{test1}{test1}
\umltrans[recursive=-160|-120|2.5cm, recursive direction=left to bottom, arg={op
   3}, pos=1.5]{test1}{test1}
\umltrans[recursive=-20|-60|2.5cm, recursive direction=right to bottom, arg={op4},
    pos=1.5]{test1}{test1}
\umlbasicstate[y=-8, name=test2, fill=white]{test2}
\umltrans[recursive=-160|-120|2.5cm, recursive direction=left to bottom, arg={op
   5}, pos=1.5]{test2}{test2}
\umltrans{test1}{test2}
\umlstatefinal[x=3, y=-7.75, name=Bfinal]
\umltrans{test2}{Bfinal}
\end{umlstate}
\umlstateinitial[x=6, y=1, name=Ainit]
\umlVHtrans[anchor2=40]{Ainit}{Bgraph}
\umlstatefinal[x=6, y=-3.5, name=Afinal]
\umlHVtrans[anchor1=30]{Bgraph}{Afinal}
\umlbasicstate[x=6, y=-6, name=visu, fill=green!20]{Visualisation}
\umlHVtrans{Bfinal}{visu}
\umltrans{visu}{Afinal}
\umltrans[recursive=-20|-60|2.5cm, recursive direction=right to bottom, arg=a, pos
   =1.5]{visu}{visu}
\end{umlstate}
```


Etat global de l'objet A
graphe B
op2
op1
test1
op3
op4
Visualisation
a
test2
op5

Chapter 4

Sequence diagrams

Here is an example of sequence diagram you can draw :



Now, we will talk about elements that compose such diagrams.

4.1 To define a sequence diagram

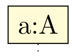
Here is the main difference from previous diagrams : to define a sequence diagram, you have to use a `umlseqdiag` environment. Its aim is to initialise some global variables and to draw the lifelines of each object in the diagram. You have to understand that commands and environments you will use to define a sequence diagrams place the elements (calls, objects, ...) automatically. We will talk about that in more details later.

4.2 To define an object

4.2.1 Types of objects

You can define an object with the `umlobject` command :

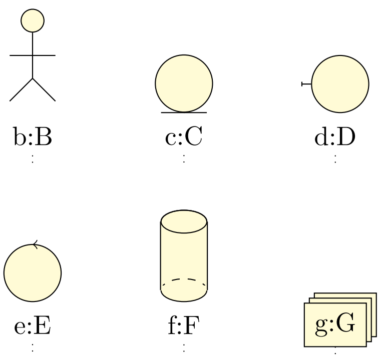
```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\end{umlseqdiag}
\end{tikzpicture}
```



The default type is a class instance. You can give the class name with the `class` option (empty by default).

The `stereo` option allows to set the type of object. It needs one of the following values : object (default value), actor, entity, boundary, control, database, multi. The last six are drawn in the following picture, from left to right and top to bottom :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlactor[class=B]{b}
\umlentity[x=2, class=C]{c}
\umlboundary[x=4, class=D]{d}
\umlcontrol[x=0, y=-2.5, class=E]{e}
\umldatabase[x=2, y=-2.5, class=F]{f}
\umlmulti[x=4, y=-2.5, class=G]{g}
\end{umlseqdiag}
\end{tikzpicture}
```



4.2.2 Automatic placement of an object

Both options `x` and `y` allows to place an object. You only have to use them if the automatic placement does not do what you need. Its behavior is the following :

- The default value of the `y` option is 0, that means the default placement of an object is at the top of the sequence diagram.
- The default value of the `x` option is the product of 4 by the value of the global counter identifying the object : for instance, for the second object defined in a diagram, the `x` option is set to 8 by default, ...

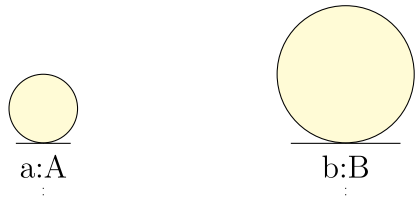
Unless the width of the object is too large, a shift of 4 is enough. If not, you use the `x` option.

4.2.3 To scale an object

The `scale` option of the `umlobject` command allows to scale an object, its symbol and its font size :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A, stereo=entity]{a}
\umlobject[x=4, scale=2, class=B, stereo=
    entity]{b}
\end{umlseqdiag}
\end{tikzpicture}
```


```
\tikzumlset{font=\large}
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A, stereo=entity]{a}
\umlobject[x=4, scale=2, class=B, stereo=
    entity]{b}
\end{umlseqdiag}
\end{tikzpicture}
```



4.3 To define a function call

Function calls are the core of sequence diagrams. Then, we need a motor either smart enough to propose a satisfacting default behavior, either easy enough to parametrize.

From a technical point of view, and I open here a parenthesis, there are two ways to implement function calls :

1. Either we use the nodal matrix structure of TikZ . The advantage is to work on a pre computed nodal grid and then to place elements of a sequence diagram easily (and fast for compilation) with exactly one counter.

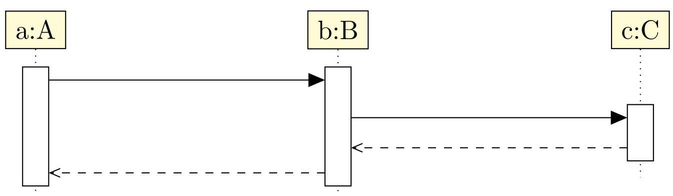
2. Either we use an automatical positioning of nodes with a set of coordinates, here the time instant, that allows total freedom for the user and make its work easier.

I chose the second way, to keep the philosophy used to implement the other diagrams in this package. Indeed, if the lack of a grid needs a more accurate computation core, and as a result more compilation time, you can define most of the elements very easily, such as constructor calls, drawn according to the standard. That is different from others UML softwares I used before. I close the parenthesis.

4.3.1 Basic / recursive calls

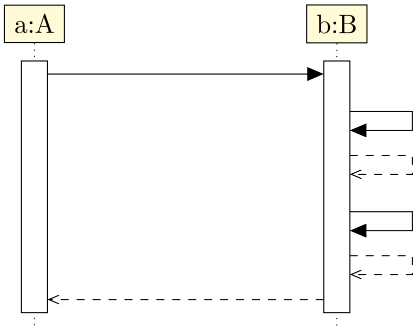
You can define a function call with the `umlcall` environment. Of course, you can define `umlcall` environments in other ones :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlobject[class=B]{b}
\umlobject[class=C]{c}
\begin{umlcall}{a}{b}
\begin{umlcall}{b}{c}
\end{umlcall}
\end{umlcall}
\end{umlseqdiag}
\end{tikzpicture}
```



You have to give the name of the source object and the name of the destination object. If you give the same name as source and destination, you define a recursive call. In this case, you may prefer use an alias, the `umlcallself` environment :

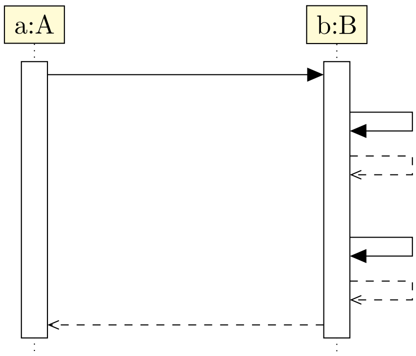
```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlobject[class=B]{b}
\begin{umlcall}{a}{b}
\begin{umlcall}{b}{b}
\end{umlcall}
\begin{umlcallself}{b}
\end{umlcallself}
\end{umlcall}
\end{umlseqdiag}
\end{tikzpicture}
```



4.3.2 To place a call

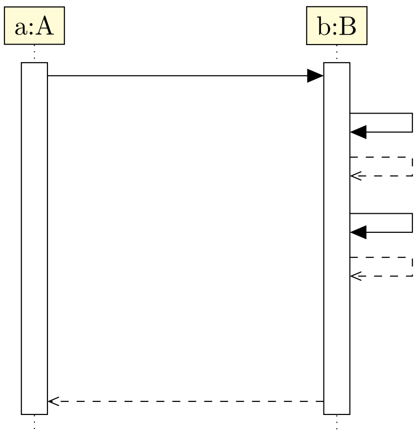
The `dt` option allows to place a function call on a lifeline, relatively to the last call drawn on this lifeline. It has no default value. Its unit is ex. The default behavior is to shift the call you define to avoid overwriting between to consecutive calls :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlobject[class=B]{b}
\begin{umlcall}{a}{b}
\begin{umlcall}{b}{b}
\end{umlcall}
\begin{umlcallself}[dt=5]{b}
\end{umlcallself}
\end{umlcall}
\end{umlseqdiag}
\end{tikzpicture}
```



You can also set spaces for recursive calls with the `padding` option. It set the space just below the recursive call :

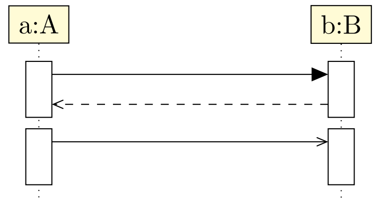
```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlobject[class=B]{b}
\begin{umlcall}[padding=10]{a}{b}
\begin{umlcall}{b}{b}
\end{umlcall}
\begin{umlcallself}{b}
\end{umlcallself}
\end{umlcall}
\end{umlseqdiag}
\end{tikzpicture}
```



4.3.3 Synchron / asynchron calls

The `type` option allows to tell if the call is synchron (default value) or asynchron :

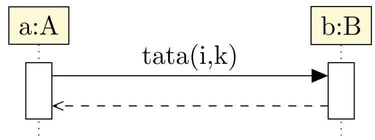
```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlobject[class=B]{b}
\begin{umlcall}[type=synchron]{a}{b}
\end{umlcall}
\begin{umlcall}[type=asynchron]{a}{b}
\end{umlcall}
\end{umlseqdiag}
\end{tikzpicture}
```



4.3.4 Operation, arguments and return value

You can give the function name in a call and its arguments with the `op` option :

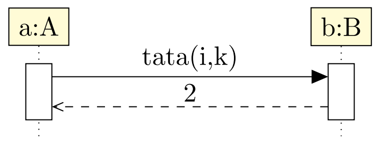
```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlobject[class=B]{b}
\begin{umlcall}[op={tata(i,k)}]{a}{b}
\end{umlcall}
\end{umlseqdiag}
\end{tikzpicture}
```



⚠ Beware of the braces, so as to the comma between i and k is deactivated as an option delimiter. WIthout them, there will be a compilation error.

You can also set the return value with the `return` option, with the same warning :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlobject[class=B]{b}
\begin{umlcall}[op={tata(i,k)}, return
    =2]{a}{b}
\end{umlcall}
\end{umlseqdiag}
\end{tikzpicture}
```

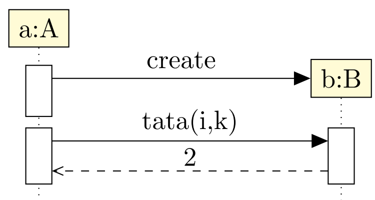


4.3.5 To define a constructor call

Constructor calls are special function calls, insofar as they build a new object. They are not messages between two lifelines, but between a lifeline and an object.

To define a constructor call, you can use the `umlcreatecall` command :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlcreatecall[class=B]{a}{b}
\begin{umlcall}[op={tata(i,k)}, return
    =2]{a}{b}
\end{umlcall}
\end{umlseqdiag}
\end{tikzpicture}
```



You can notice that everything behave normally after a constructor call.

As an object builder, the `umlcreatecall` command has `class`, `stereo` and `x` options.
As a function call, it has the `dt` option.

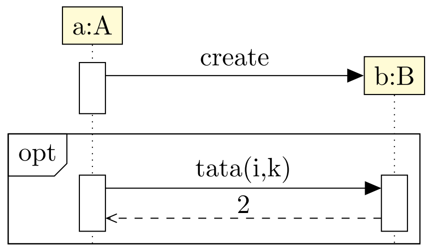
4.3.6 To name a call

The `name` option allows to give a name for a function call. It is not useful actually, insofar as this option was added for the definition of combined fragments, but as you will see, combined fragment does not use this feature. Maybe this option will be used for future developments of the package.

4.4 To define a combined fragment

Combined fragments are the second family of elements in a sequence diagram, with the function calls. You can define them with the `umlfragment` environment :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlcreatecall[class=B]{a}{b}
\begin{umlfragment}
\begin{umlcall}[op={tata(i,k)}, dt=7,
    return=2]{a}{b}
\end{umlcall}
\end{umlfragment}
\end{umlseqdiag}
\end{tikzpicture}
```

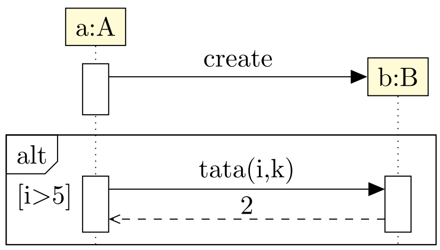


4.4.1 Informations of a fragment

The `type` option allows to set the keyword on the top left corner : opt, alt, loop, par, assert, . . . The default value is opt.

The `label` option allows to set information such as the condition for a opt fragment :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlcreatecall[class=B]{a}{b}
\begin{umlfragment}[type=alt, label=i>5,
    inner xsep=2]
\begin{umlcall}[op={tata(i,k)}, dt=7,
    return=2]{a}{b}
\end{umlcall}
\end{umlfragment}
\end{umlseqdiag}
\end{tikzpicture}
```

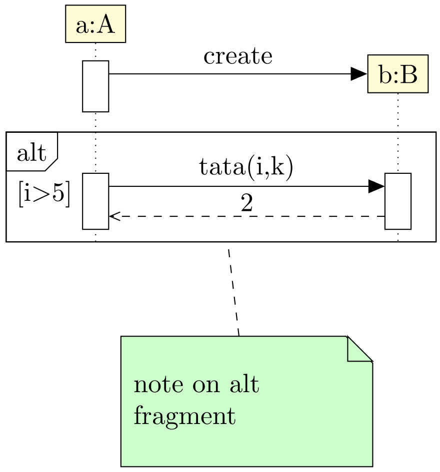


The `inner xsep` option allows to shift type and label to the left. The default value is 1 and its unit is ex.

4.4.2 Name of a fragment

You can give a name to a combined fragment with the `name` option. It can be useful when you want to attach a note on a fragment :

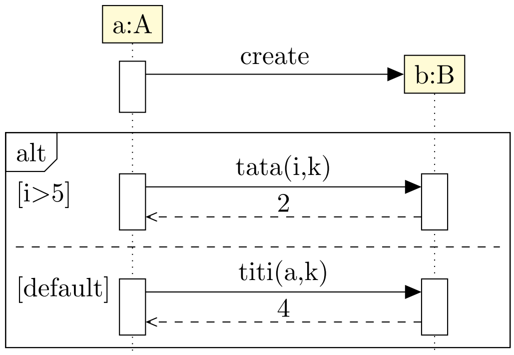
```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlcreatecall[class=B]{a}{b}
\begin{umlfragment}[type=alt, label=i>5,
   name=alt, inner xsep=2]
\begin{umlcall}[op={tata(i,k)}, dt=7,
   return=2]{a}{b}
\end{umlcall}
\end{umlfragment}
\umlnote[x=2, y=-5]{alt}{note on alt
   fragment}
\end{umlseqdiag}
\end{tikzpicture}
```



4.4.3 To define regions of a fragment

Let's take a alt fragment. It represents a switch-case instruction block. To represent each case, you need to set regions in the fragment. For this purpose, you can use the `umlfpart` command :

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlobject[class=A]{a}
\umlcreatecall[class=B]{a}{b}
\begin{umlfragment}[type=alt, label=i>5,
   inner xsep=5]
\begin{umlcall}[op={tata(i,k)}, dt=7,
   return=2]{a}{b}
\end{umlcall}
\umlfpart[default]
\begin{umlcall}[op={titi(a,k)}, return
   =4]{a}{b}
\end{umlcall}
\end{umlfragment}
\end{umlseqdiag}
\end{tikzpicture}
```



4.5 To change preferences

Thanks to the `tikzumlset` command, you can set colors for calls, fragments and objects :

text : allows to set the text color (=black by default),

draw : allows to set the color od edges and arrows (=black by default),

fill object : allows to set the background color of objects (=yellow !20 by default),

fill call : allows to set the background color for calls (=white by default),

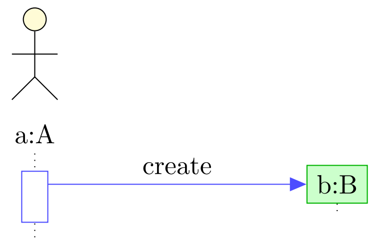
fill fragment : allows to set the background color for fragments (=white by default),

font : allows to set the font style (=\small by default).

You can also use the options `text`, `draw` and `fill` on a particular element, as in the example of introduction.

There is an exception : `umlcreatecall`. The options `text`, `draw` and `fill` set the colors of the call, whereas options `text obj`, `draw obj` and `fill obj` set the colors of the object.

```
\begin{tikzpicture}
\begin{umlseqdiag}
\umlactor[class=A]{a}
\umlcreatecall[class=B, draw obj=green!70!black,
    fill obj=green!20, draw=blue!70]{a}{b}
\end{umlseqdiag}
\end{tikzpicture}
```



4.6 Examples

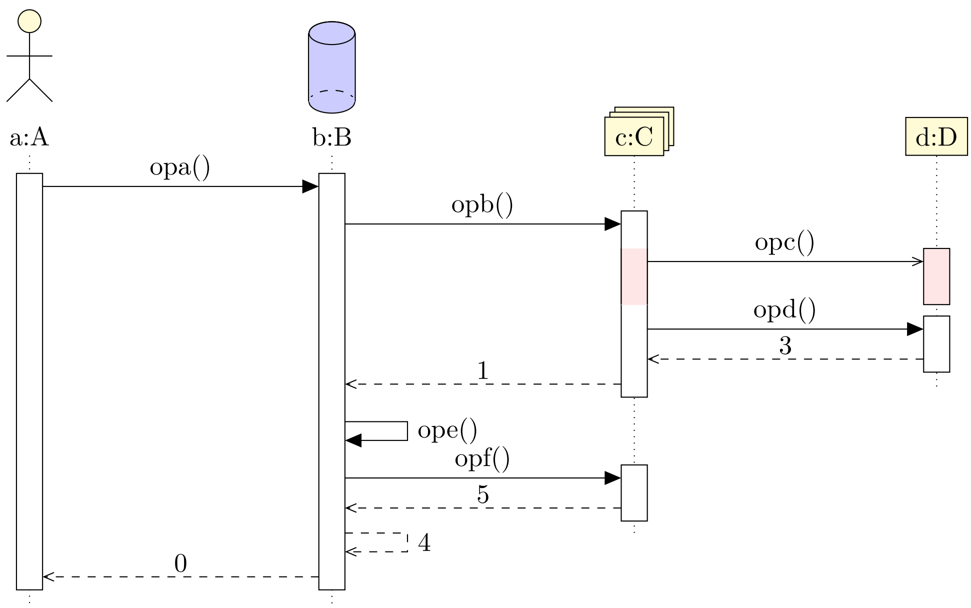
4.6.1 Example from introduction, step by step

Definition of objects

```
\begin{umlseqdiag}
\umlactor[class=A]{a}
\umldatabase[class=B, fill=blue!20]{b}
\umlmulti[class=C]{c}
\umlobject[class=D]{d}
\end{umlseqdiag}
```

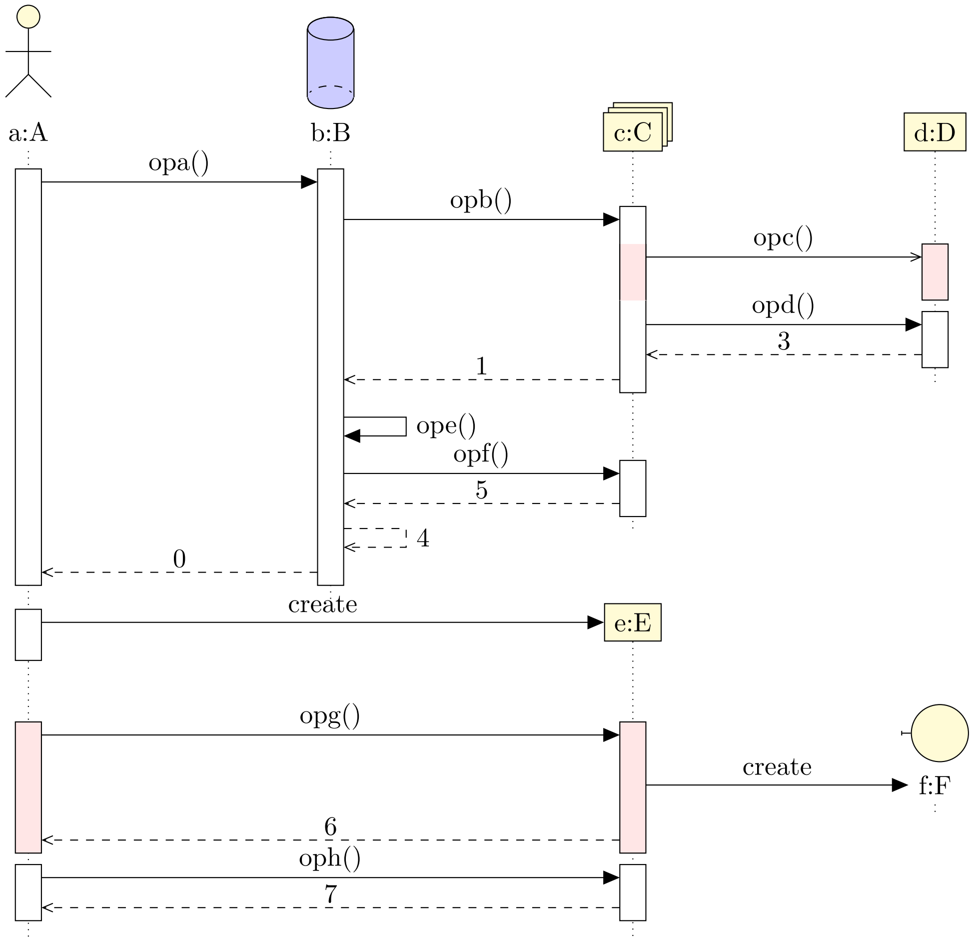
Definition of the call opa and its components

```
\begin{umlseqdiag}
\umlactor[class=A]{a}
\umldatabase[class=B, fill=blue!20]{b}
\umlmulti[class=C]{c}
\umlobject[class=D]{d}
\begin{umlcall}[op=opa(), type=synchron, return=0]{a}{b}
\begin{umlcall}[op=opb(), type=synchron, return=1]{b}{c}
\begin{umlcall}[op=opc(), type=asynchron, fill=red!10]{c}{d}
\end{umlcall}
\begin{umlcall}[op=opd(), type=synchron, return=3]{c}{d}
\end{umlcall}
\end{umlcall}
\begin{umlcallself}[op=ope(), type=synchron, return=4]{b}
\begin{umlcall}[op=opf(), type=synchron, return=5]{b}{c}
\end{umlcall}
\end{umlcallself}
\end{umlcall}
\end{umlseqdiag}
```



Definition of the calls following the construction of E

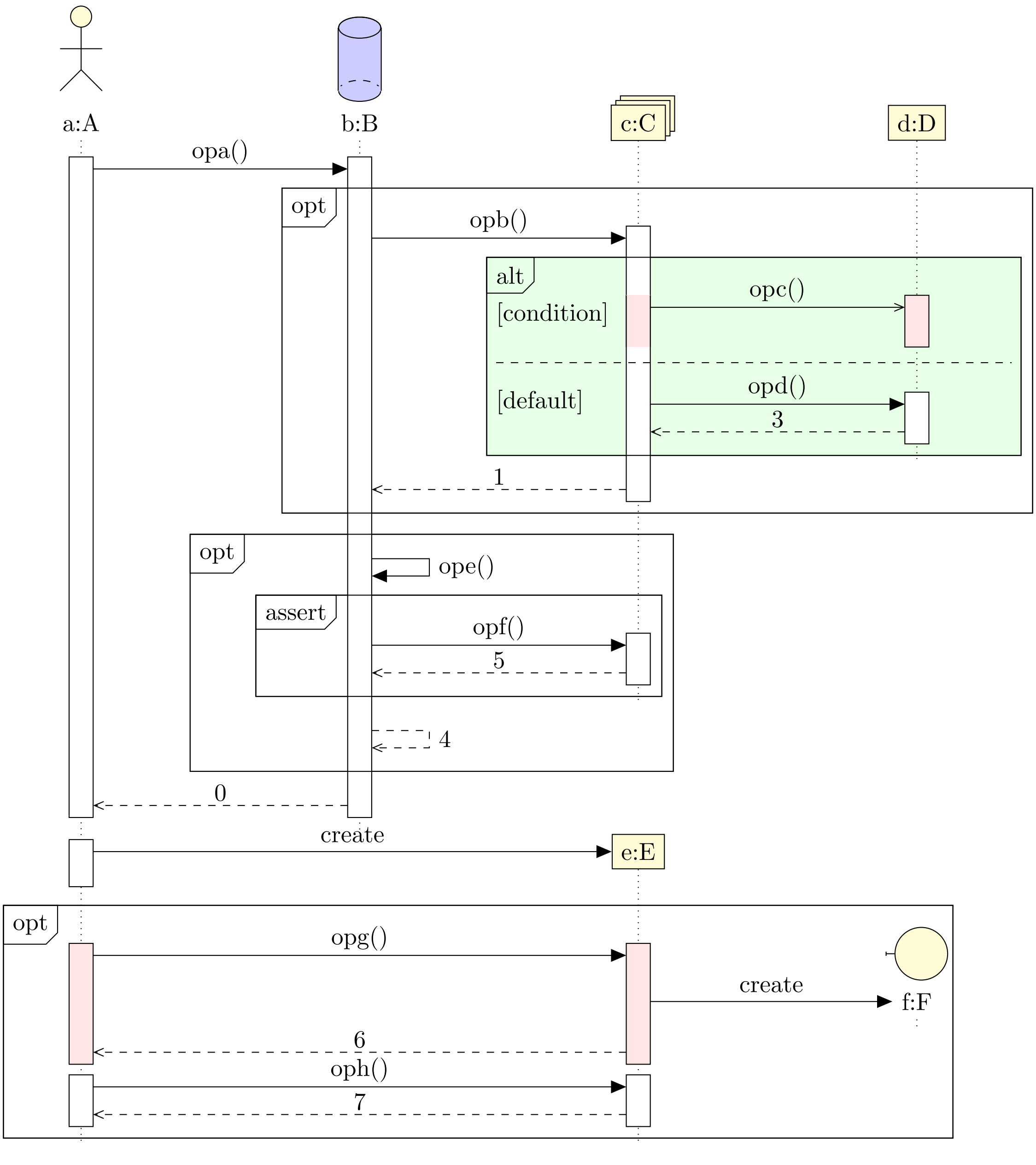
```
\begin{umlseqdiag}
\umlactor[class=A]{a}
\umldatabase[class=B, fill=blue!20]{b}
\umlmulti[class=C]{c}
\umlobject[class=D]{d}
\begin{umlcall}[op=opa(), type=synchron, return=0]{a}{b}
\begin{umlcall}[op=opb(), type=synchron, return=1]{b}{c}
\begin{umlcall}[op=opc(), type=asynchron, fill=red!10]{c}{d}
\end{umlcall}
\begin{umlcall}[op=opd(), type=synchron, return=3]{c}{d}
\end{umlcall}
\end{umlcall}
\begin{umlcallself}[op=ope(), type=synchron, return=4]{b}
\begin{umlcall}[op=opf(), type=synchron, return=5]{b}{c}
\end{umlcall}
\end{umlcallself}
\end{umlcall}
\umlcreatecall[class=E, x=8]{a}{e}
\begin{umlcall}[op=opg(), name=test, type=synchron, return=6, dt=7, fill=red!10]{a
    }{e}
\umlcreatecall[class=F, stereo=boundary, x=12]{e}{f}
\end{umlcall}
\begin{umlcall}[op=oph(), type=synchron, return=7]{a}{e}
\end{umlcall}
\end{umlseqdiag}
```



Definition of fragments

```
\begin{umlseqdiag}
\umlactor[class=A]{a}
\umldatabase[class=B, fill=blue!20]{b}
\umlmulti[class=C]{c}
\umlobject[class=D]{d}
\begin{umlcall}[op=opa(), type=synchron, return=0]{a}{b}
\begin{umlfragment}
\begin{umlcall}[op=opb(), type=synchron, return=1]{b}{c}
\begin{umlfragment}[type=alt, label=condition, inner xsep=8, fill=green!10]
\begin{umlcall}[op=opc(), type=asynchron, fill=red!10]{c}{d}
\end{umlcall}
\umlfpart[default]
\begin{umlcall}[op=opd(), type=synchron, return=3]{c}{d}
\end{umlcall}
\end{umlfragment}
\end{umlcall}
\end{umlfragment}
\begin{umlfragment}
\begin{umlcallself}[op=ope(), type=synchron, return=4]{b}
\begin{umlfragment}[type=assert]
\begin{umlcall}[op=opf(), type=synchron, return=5]{b}{c}
\end{umlcall}
\end{umlfragment}
\end{umlcallself}
\end{umlfragment}
```

```
\end{umlcall}
\umlcreatecall[class=E, x=8]{a}{e}
\begin{umlfragment}
\begin{umlcall}[op=opg(), name=test, type=synchron, return=6, dt=7, fill=red!10]{a
    }{e}
\umlcreatecall[class=F, stereo=boundary, x=12]{e}{f}
\end{umlcall}
\begin{umlcall}[op=oph(), type=synchron, return=7]{a}{e}
\end{umlcall}
\end{umlfragment}
\end{umlseqdiag}
```



4.7 Known bugs and perspectives

1. When you define a fragment on a set of calls just after a constructor call, the automatic shift does not work. You have to use the `dt` with a value greather than 7 to the first call inside the fragment.

2. The automatic placement of objects with a multiple of 4 is not very convenient. A shift of 4 relatively to the last object drawn should be better.

3. You can not give arguments to constructor calls.

4. You can not force the drawing of the activity area of a ”non working” object.